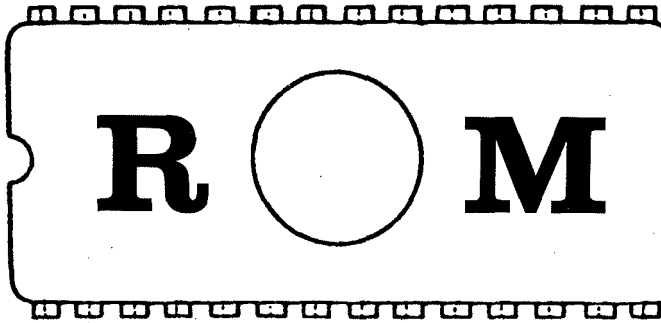
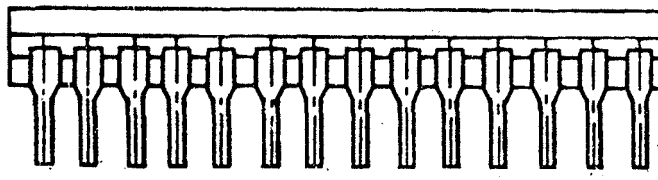
II, II+, IIe, IIc



Listing

für Applesoft-BASIC-Interpreter

von Matthias Buck



RÖCKRATH
MIKROCOMPUTER

Matthias Buck: Apple-II-ROM-Listing
Best.-Nr. 06-010-8

2. Auflage, November 1984

(c) 1984 by

RÖCKRATH
MICROCOMPUTER
Noppiusstr. 19
5100 Aachen

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeinem Verfahren reproduziert oder in EDV-Anlagen verarbeitet werden.

Alle in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt und dürfen nicht gewerblich genutzt werden.

Alle Angaben in diesem Buch wurden mit größtmöglicher Sorgfalt erarbeitet bzw. zusammengestellt. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag und der Autor sehen sich deshalb gezwungen, darauf hinzuweisen, daß sie weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung eventueller Fehler sind Autor und Verlag jederzeit dankbar.

INHALTSVERZEICHNIS

Einführung in den Applesoft-Interpreter	4
Speicheraufteilung	4
Zeropage-Adressen	5
Sonstige Systemadressen	7
Stackorganisation	8
Variablenverwaltung	9
Stringverwaltung	10
Beschreibung einiger Routinen	11
ROM-Tabellen	15
Kommentiertes Listing der ROM-Routinen	19
Alphabetische Übersicht über die ROM-Routinen	116

ANHANG

Änderungen bei den Geräteversionen IIe und IIc	119

Einführung in den Applesoft-Interpreter

Nach dem Einschalten des Computers führt der Monitor eine Resetroutine durch und veranlasst das Booten von DOS, falls ein Disk-Drive vorhanden ist. Die Kaltstartroutine initialisiert sodann die notwendigen Zero-Page-Adressen und den Stack, und führt schliesslich einen Sprung zur Warmstart-Adresse $D43C durch. Dies ist nun der Grundzustand.

Die Warmstart-Routine erwartet nun eine Eingabezeile. Dies geschieht über die Monitor-Routine GETLN. Bei Rückkehr von dieser Routine steht der eingegebene Text im Textpuffer ab $0200 und kann ausgewertet werden.

War das erste Zeichen eine Ziffer, so wird die eingegebene Zeile als neue Programmzeile interpretiert. Die Zeile wird dann kodiert, d.h. erkannte Keywords werden durch die zugehörigen Tokens ersetzt. Im bestehenden Programmtext wird an der richtigen Stelle Platz für die neue Zeile geschaffen, und diese dort eingefügt. Nachdem die Linkadressen in Programm neu berechnet worden sind, erfolgt wieder ein Sprung zum Warmstart.

War das erste Zeichen keine Ziffer, so wird die eingegebene Zeile als Befehlszeile aufgefasst. Es werden ebenfalls die Befehlswörter kodiert, es erfolgt dann aber ein Sprung zur Interpreter-Schleife.
Wird ein Programm durch RUN gestartet, so wird es ebenfalls von der Interpreter-Schleife ausgeführt.
Die Interpreter-Schleife bestimmt aufgrund des Tokens die Adresse der zugehörigen Routine und ruft diese auf. Nach Rückkehr in die Interpreter-Schleife wird das nächste Token ausgewertet usw. bis die Befehlszeile bzw. das Programm ganz abgearbeitet ist.
In diesem Fall erfolgt ein Sprung zurück zur Warmstart-Routine.

Es folgt nun eine Beschreibung wesentlicher Routinen und Strukturen, die das Verständnis des kommentierten Interpreter-Listings erleichtern sollen.

Speicheraufteilung

0000 - 00FF	Zeropage, enthält Flags und Adressen für den Interpreter
0100 - 01FF	Stack
0200 - 02FF	Eingabepuffer, hier legt GETLN die Eingabezeile ab
0300 - 03CF	frei
03D0 - 03FF	zusätzliche Vektoren, vor allem für DOS
0400 - 07FF	Bildschirmspeicher für TEXT und LORES-Grafik, Page1 ausserdem noch 64 Byte für die Steckkarten
0800 - BFFF	freies RAM, das sich wie folgt aufteilt:

Programmtext, Zeiger auf dessen Anfang = 67,68
einfache Variablen, Anfangszeiger = 69,6A
Feldvariablen, Anfangszeiger = 6B,6C
unbenutzter Bereich, Anfangszeiger = 6D,6E
Stringbereich, Anfangszeiger = 6F,70
Ende +1 des von Basic erreichbaren Speichers = 73,74

ausserdem noch folgende Bereiche:

0800 - 0BFF	TEXT und LORES-Grafik, Page2
2000 - 3FFF	HIRES-Grafik, Page1
4000 - 5FFF	HIRES-Grafik, Page2
9600 - BFFF	wird vom DOS belegt, falls vorhanden

C000 - C7FF	I/O-Bereich, aufgeteilt auf die Slots und internes I/O
C800 - CFFF	Bereich von 2K, der jeder Karte zur Verfügung steht
D000 - F7FF	Applesoft-Interpreter
F800 - FFFF	Monitor

Liste der Zeropage-Adressen

00 - 02	JMP zur Warmstart-Routine
03 - 05	JMP zur String-Druckroutine
06 - 09	frei
0A - 0C	JMP zur USR-Routine (nach Kaltstart JMP E199)
VFR HLP 0D,0E	Hilfsregister zum Vergleich einzelner Zeichen
0F	Anzahl der Dimensionen des letzten Feldes Länge einer neuen Basic-Zeile Token-Zähler bei Befehlskodierung
10	DIM-Flag
11	String-Flag, (FF falls String)
12	Integer-Flag (80 falls Integer, sonst 00)
13	Garbage Collection-Flag DATA-Flag bei Befehlskodierung
14	Flag für Variablenverwaltung: 00 normale Variablen-Behandlung 40 Feldvariable (bei STORE, ohne Indices) 80 keine Integer-Variablen erlaubt
15	Eingabe-Flag: 00 = INPUT, 40 = GET, 98 = READ
16	Vergleichs-Flag bei Vergleichsoperationen Hilfsregister für TAN-Berechnung
17 - 19	frei
1A,1B	Zeiger bei Grafikroutinen
1C	Farbmaske (evtl. invertiert)
1D	Schrittzähler beim Zeichnen einer Linie
1E,1F	frei
20	Bildschirmfenster, linker Rand
34 21	Breite des Bildschirmfensters
35 22,23 36	oberer / unterer Rand des Bildschirmfensters
37 24,25 38	horizontale / vertikale Cursorposition
39 26,27 40	Zeiger auf Anfang einer Grafikzeile
28 - 2B	Monitor-Register
2C,2D	Koordinate bei Blockgrafik
2E,2F	Monitor-Register
30	Bitmaske für Punkt innerhalb eine Bytes (HGR)
31	Monitor-Register
32	INVERSE-Flag
33	Register für Prompt-Zeichen
34,35	Monitor-Register
36,37	Vektor für Ausgaberoutine im Monitor
38,39	Vektor für Eingaberoutine im Monitor
3A,3B	Monitor-Register für Programmzeiger
3C,3D	Anfangsadresse für Cassetten-Routinen
3E,3F	Endadresse für Cassetten-Routinen
40 - 45	Monitor-Register
46 - 49	Monitor-Ablage für A,X,Y,Status,Stackpointer
4A,4B	DOS-Register
4C,4D	frei
4E,4F	Monitor-Zufallszahl
50,51	Register für Integerzahlen
52	Stackpointer des Deskriptoren-Stack
53,54	Zeiger auf obersten Deskriptor im D.-Stack
55 - 5D	Deskriptoren-Stack für max. 3 Deskriptoren
Zeiger 1 5E,5F	allgemeiner Zeiger
Zeiger 2 60,61	allgemeiner Zeiger
Hilfsreg 62 - 65	Hilfsregister für Multiplikation / Division
66	wird bei Division verändert
67,68	Anfang des Programmtextes, normal $0801
69,6A	LOMEM, = Anfang der einfachen Variablen
6B,6C	Anfang der Feldvariablen
6D,6E	Feldvariablen-Ende +1

6F,70	Anfang des Stringbereichs
71,72	Zeiger bei Stringverwaltung
73,74	HIMEM, = Ende +1 des verfügbaren RAM
75,76	aktuelle Programmzeilennummer
77,78	Programmzeile, bei der Unterbrechung erfolgte
79,7A	Zeiger auf letzten Befehl (für ONERR etc.)
	7A = FF, falls im Direktmodus
7B,7C	Zeilennummer des aktuellen DATA-Statements
7D,7E	Zeiger auf dieses DATA-Statement
7F,80	Zeiger bei Eingaberoutine
81,82	Register für letzten Variablen-Namen
83,84	Zeiger auf zuletzt benutzte Variable
85,86	gemerkter Variablenzeiger bei LET, NEXT, etc.
87,88	gemerkter CHRGET-Zeiger bei Eingabeoperation
87	Register für Priorität bei num. Ausdrücken
89	Vergleichs-Flag
8A - 8E	FAC5 (wird bei Potenzierungsroutine benützt)
8A,8B	Zeiger auf FN-Definitions-Variable
	Zeiger auf Top-String bei Garbage Collection
8C,8D	Zeiger auf gültigen String-Deskriptor
8F	Länge einer Variablen im Speicher (3 oder 7)
90 - 92	JMP-Befehl bei Funktionsaufruf
93 - 97	FAC3
94,95	Zeiger auf 1. Element einer Feldvariablen
96,97	altes Blockende +1 bei Verschiebe-Routine
98 - 9C	FAC4
99	Hilfsregister (z.B. bei FAC1 —> String)
9A	Hilfsregister bei String <--> FP-Konstante
9B,9C	allgemeiner Zeiger
9D - A1	FAC1
A2	Vorzeichen von FAC1
A3	Hilfsregister (z.B. bei Polynomauswertung)
A4	Hilfsregister für Mantissenverschiebung
A5 - A9	FAC2
AA	Vorzeichen von FAC2
AB,AC	Zeiger bei Stringverwaltung
AB	Kombi-Vorzeichen (verknüpftes Vorzeichen)
AC	Rundungsstelle für FAC1
AD	Hilfsregister bei Befehlskodierung
AD,AE	Platzbedarf bei Verwaltung von Feldvariablen
	gemerkter CHRGET-Zeiger bei VAL-Routine
AF,B0	Programmtext-Ende +1
B1 - C8	CHRGET-Routine
B8,B9	CHRGET-Zeiger
C9 - CD	Register für Zufallszahl bei RND
CE,CF	frei
D0 - D5	Hilfsregister für HGR-Routinen
D6	Autostart-Flag
D7	frei
D8	ONERR-Flag
D9	frei
DA,DB	Programmzeile, die einen Fehler verursachte
DC,DD	Zeiger auf den Befehl bei dem Fehler auftrat
DE	Kode des aufgetretenen Fehlers
DF	gemerkter Stackpointer bei Fehlerbehandlung
E0,E1	X-Koordinate bei HGR-Routinen
E2	Y-Koordinate bei HGR-Routinen
E3	frei
E4	Farbmaske entsprechend letztem HCOLOR-Befehl
E5	Spaltenindex eines Punktes in HGR-Routinen
E6	Page-Flag (20 für Page 1, 40 für Page 2)

E7	SCALE-Wert
E8,E9	Zeiger auf Anfang des Shape-Tables
EA	Collision Counter bei DRAW / XDRAW
EB - EF	frei
F0	Hilfsregister für Blockgrafik
F1	SPEED-Wert
F2	TRACE-Flag
F3	FLASH-Flag, =40 bei FLASH, sonst 00
F4,F5	Zeiger auf gültiges ONERR-Statement
F6,F7	Programmzeile mit gültigem ONERR-Statement
F8	gemerkter Stackpointer
F9	ROT-Wert
FA - FE	frei Fac 6
FF	wird bei STR$ verändert

Vektoren in Page 3

3D0	Sprung zum DOS-Warmstart
3D3	Sprung zum DOS-Kaltstart
3D6	Sprung zur DOS-Filemanager-Routine
3D9	Sprung zur DOS-RWTS-Routine
3DC	Hilfsroutine für Filemanager-Parameter
3E3	Hilfsroutine für RWTS-Parameter
3FA	Sprung zur Routine, die DOS-I/O-Kanäle setzt
3EF	Sprung zur Routine, die BRK-Befehle eines Maschinenprogrammes bearbeitet (normal $FA59)
3F2	Sprungadresse bei RESET (für AUTOSTART ROM) = Soft Entry Vector
3F4	Power-up Byte, dient zur Unterscheidung von Tastatur-RESET und Power-on-Reset
3F5	Sprung zur &-Routine. Der Applesoft-Befehl "&" bewirkt einen Sprung hierher, von hier dann zur Routine. (JMP $FF58 falls nicht benutzt)
3F8	Sprung zur Monitor-User-Routine. (wird bewirkt durch Eingabe von Ctrl Y)
3FB	Sprung zu der Routine, die Non-Maskable Interrupts bearbeitet
3FE	Adresse der Routine, die Interrupt Requests bearbeitet. Diese Interrupts können durch den SEI-Befehl verhindert werden.

Interne I/O-Adressen

C000	Tastatur-Register
C010	bewirkt Löschung des Tastatur-Registers
C020	Flipflop für Cassettenausgang umschalten
C030	Flipflop für Lautprecher umschalten
C040	Utility-Strobe am Game-Port auslösen
C050/C051	GRAPHICS/TEXT-Modus setzen
C052/C053	NOMIX/MIX-Modus setzen
C054/C055	PAGE1/PAGE2 darstellen
C056/C057	LORES/HIRES Grafikmodus setzen
C058-C05F	Register für TTL-Ausgänge AN0 - AN3
C060	Cassetteneingangs-Register (Bit7)
C061-C063	Register für TTL-Eingänge PB0 - PB2
C064-C067	Register für Analog-Eingänge (Bit7)
C068-C06F	gleich wie C060 - C067
C070	Monoflops der Analogeingänge rücksetzen

Stack-Organisation

Der Stack ist ein Speicherbereich von 256 Byte, der durch die Assemblerbefehle PHA/PLA und PHP/PLP eine sehr elegante Möglichkeit, Daten zwischenzeitlich abzulegen, eröffnet. Im einzelnen wird er wie folgt benutzt:

Ablage der Rücksprung-Adresse bei Unterprogramm-Aufruf durch JSR

Kurzzeitiges Retten von Registern, Flags etc. in Assemblerroutinen

Ablage der RETURN-Daten bei einer Basic-Anweisung GOSUB:
- CHRGET-Zeiger auf GOSUB-Statement (2 Byte)
- Zeilennummer des GOSUB-Statements (2 Byte)
- B0 als Kennung für RETURN-Daten (1 Byte), insgesamt also 5 Byte

Ablage der Schleifen-Parameter einer FOR/NEXT-Schleife:
- CHRGET-Zeiger auf 1. Anweisung nach dem FOR-Statement (2 Byte)
- Zeilennummer der zugehörigen Programmzeile (2 Byte)
- TO-Wert im Fliesskomma-Format (5 Byte)
- STEP-Wert im Fliesskomma-Format + Vorzeichenbyte (6 Byte)
- Zeiger auf die Laufvariable (2 Byte)
- 81 als Kennung für FOR/NEXT-Daten (1 Byte), insgesamt also 18 Byte

Ablage der Daten für eine offene Operation bei der Auswertung von arithmetischen Ausdrücken:
- Adresse für Rücksprung nach $DDDA wenn Operation fertig (2 Byte)
- Sprungadresse zur Routine, die die Operation ausführt (2 Byte)
- Operand im Fliesskomma-Format + Vorzeichenbyte (6 Byte)
- Vergleichs-Flag, enthält Vergleichskode (1 Byte)
- Prioritäts-Flag (1 Byte), insgesamt also 12 Byte

Übergabe von Parametern an die LEFT$/MID$/RIGHT$-Routinen:
- Zeiger auf den Deskriptor des Original-Strings
- 1. Parameter, insgesamt 3 Byte

Übergabe der Indices für die einzelnen Dimensionen bei der Auswertung von Feldvariablen

Die Verwaltung des Stack

Der Stack-Pointer ist ein Register in der 6502 CPU, welches als Zeiger auf die nächste zu beschreibende Stackposition dient. Bei Ausführung eines PHA wird das Byte in diese Position eingetragen, und der Stackpointer auf die nächste Position gesetzt. Umgekehrt wird bei PLA zunächst der Stack-Pointer eine Position zurückgesetzt, und das dort stehende Byte geholt. Dabei wird der Stack von oben nach unten gefüllt. Bei Stackinitialisierung wird der Stackpointer auf F8 gesetzt. Das erste Byte wird also nach $01F8 eingetragen, das nächste nach $01F7 etc.

In der Routine ab $D3D6 wird geprüft, ob noch mindestens 54 Byte auf dem Stack frei bleiben. Damit ist gewährleistet, daß bei maximaler Unterprogrammverschachtelung der Stack nicht überläuft. Außerdem legt die Routine ab $ED34 das String-Abbild der Fliesskommazahl im unteren Stack-Bereich (ab $0100 bzw. $00FF) ab.

Verwaltung der Variablen

Die Variablen stehen im Speicher zwischen den Zeigern 69,6A (=LOMEM) und 6D,6E. Durch LOMEM: lässt sich dieser Bereich zwischen Programmtextende und HIMEM einstellen, ansonsten folgt er direkt dem Programmtext.
Zunächst folgen die einfachen Variablen, hinter denen dann ab (6B,6C) die Feldvariablen stehen. Innerhalb dieser beiden Bereiche gibt es keine Gliederung, sondern FP-Variable (FP = Floating Point = Fliesskomma), Integer-Variable und String-Variable werden in der Reihenfolge ihrer Definition abgelegt. Wird nun z.B. eine neue einfache Variable definiert, so werden die Feldvariablen verschoben, und die neue Variable am Ende der einfachen Variablen eingefügt.
Wird dagegen CLEAR durchgeführt, so wird nicht der Speicher gelöscht, sondern es werden lediglich die Zeiger auf die Bereichsenden auf die Variablen-Anfangsadresse (LOMEM) gesetzt.

Einfache Variablen belegen immer 7 Byte. Die ersten beiden enthalten den Variablennamen, wobei in Bit7 der Variablentyp verschlüsselt ist:
- FP: Name1 (Bit7=0); Name2 (Bit7=0); Exponent; Mantissenbytes 1 - 4
- Integer: Name1 (Bit7=1); Name2 (Bit7=1); HByte; LByte; 3 Bytes = 00
- String: Name1 (Bit7=0); Name2 (Bit7=1); 3 Deskriptorbytes; 2 Bytes = 00

Die Stringvariable enthält also nicht den String selbst, sondern den Deskriptor, der die Stringlänge angibt, und wo der eigentliche String im Speicher steht. (siehe Abschnitt über die Stringverwaltung)

Feldvariable besitzen folgende Struktur:
Variablenname (2 Byte, enthält ebenfalls den Variablentyp in den Bits 7)
Länge der Feldvariablen (LByte, HByte)
Anzahl der Dimensionen (1 Byte)
Dimensionierung des letzten Index (HByte, LByte)
.....
Dimensionierung des ersten Index (HByte, LByte)
Elemente in der Folge (0,0,..,0), (1,0,..,0),.., (n,0,..,0), (0,1,..,0) etc
d.h. der erste Index wächst am schnellsten, der letzte am langsamsten.
Die Elemente haben den gleichen Inhalt wie einfache Variable, bis auf die überflüssigen Nullen. FP/Integer/String-Elemente sind also 5/2/3 Byte lang.

Variablen werden definiert, sobald ihnen ein Wert zugewiesen wird.
Dies gilt auch für Felder. Eine Zuweisung auf ein Element eines bisher nicht dimensionierten Feldes bewirkt dessen Dimensionierung mit 11 Elementen pro Dimension, d.h. wie DIM A(10,10,..,10).
Wird nur der Wert einer nicht definierten Variable verwandt, so wird die Variable dadurch nicht neu eingerichtet, und als Wert erhält man null bzw. einen String der Länge 0.

Mit STORE und RECALL können Felder auf Cassette gespeichert werden bzw. wieder von dort geladen werden. Dabei wird nur der Name angegeben und keine Indices, wie sonst üblich. Das in diesem Fall gesetzte STORE-Flag (in $14 wird Bit6 gesetzt) bewirkt, daß trotzdem eine Feldvariable geladen bzw. abgespeichert wird.

Da der Variablentyp in die beiden Namenbytes integriert ist, kann ein Name für sechs Variable verwandt werden, ohne daß Verwechslungen möglich sind, nämlich für jeden Typ, und zwar einfach oder als Feld.

Stringverwaltung

Da Strings Zeichenketten sind, deren genauer Inhalt für die Verwaltung ohne Belang ist, werden Strings meist über sogenannte Deskriptoren gehandhabt. Ein Deskriptor beinhaltet in 3 Byte die Länge des Strings, und wo er im Speicher zu finden ist. In dieser Form wird er auch in Variablen abgelegt. Der eigentliche String steht dann entweder im Programmtext, im Stringbereich am oberen RAM-Ende, oder auch im Eingabepuffer.
Wird einer Stringvariablen ein neuer Inhalt zugewiesen, so wird einfach der Deskriptor des neuen String in die Variable eingetragen, und der neue String selbst an den Anfang des Stringbereichs angehängt, falls er nicht im Programmtext steht. Dadurch kan es geschehen, daß sich im String-Bereich immer mehr "Stringleichen" ansammeln, d.h. Strings, die keiner Variablen mehr zugewiesen sind, und unnötigerweise Speicherplatz kosten. Den Stringbereich von solchen Stringleichen zu befreien ist die Aufgabe der Garbage Collection Routine. Sie vergleicht Variablenbereich und Stringbereich und übernimmt nur aktuelle Strings. Diese Routine wird durchgeführt, sobald der Speicherplatz nicht mehr ausreicht, sowie bei der Funktion FRE(0).

Um Stringdeskriptoren handhaben zu können, obwohl sie keiner Variablen zugewiesen sind, gibt es den Deskriptoren-Stack in der Zeropage. Auf ihm haben 3 Deskriptoren Platz. Wenn der Deskriptor für einen neuen String bestimmt wurde, wird er grundsätzlich zunächst auf dem Deskriptoren-Stack abgelegt ($E42A ff), und der String in den Stringbereich übernommen, falls notwendig. Sobald der neue String seine Bestimmung gefunden hat, (z.B. einer Variablen zugewiesen wurde, oder gedruckt wurde etc.), wird der Deskriptor wieder vom Deskriptoren-Stack entfernt ($E635 ff). Wird der String selbst auch nicht mehr benötigt, wird er, sofern er ganz am Anfang des Stringbereichs steht, ebenfalls gelöscht ($E600 ff).
Für den Deskriptoren-Stack gibt es auch einen Zeiger, der auf die als nächstes zu besetzende Position zeigt. Seine Funktionsweise ist also die gleiche wie beim normalen Stack.

Umwandlung einer Fliesskommazahl in einen String

Diese Aufgabe wird von der Routine ab $ED34 wahrgenommen. Durch Vergleich mit entsprechenden Konstanten werden nach und nach die einzelnen Ziffern bestimmt, außerdem Vorzeichen und Exponent. Der String wird dabei ab $00FF (also im unteren Stackbereich!) abgelegt, falls die Routine von STR$ gerufen wurde, bzw. ab $0100, falls der Aufruf von PRINT erfolgte. Im Fall von STR$ wird der Zahlenstring auch noch in den Stringbereich kopiert, bei PRINT nicht (vgl. $E3E7 ff).

Die Garbage Collection Routine

Bei der Verwaltung der Strings kann es geschehen, daß sich im Stringbereich Strings ansammeln, die keiner Variablen mehr zugewiesen sind, String-"Müll" also (vgl. Erläuterung der Stringverwaltung). Die Garbage Collection Routine ab $E484 beseitigt diese aus dem Stringbereich.
Dazu wird zunächst der Stringbereichs-Anfang auf HIMEM gesetzt; nun gibt es also keine gültigen Strings im Stringbereich.
Es werden nun sämtliche Stringvariablen sowie der Deskriptoren-Stack durchsucht, um den String mit der höchsten Adresse zu finden, der nicht im Stringbereich steht, d.h. der noch nicht als aktuell erkannt wurde. Dieser wird schliesslich in den Stringbereich übernommen, und der zugehörige Deskriptor aktualisiert. Dieser String, im Kommentar als Topstring bezeichnet, wird in den folgenden Durchgängen also nicht mehr berücksichtigt. Nun werden erneut sämtliche Variablen untersucht, um aus den verbliebenen Strings den höchsten gültigen herauszufinden usw, bis endlich alle aktuellen Strings in den Stringbereich übernommen worden sind.
Da der Zeitaufwand für diese Routine etwa quadratisch mit der Anzahl der Strings wächst, kann es bei vielen Strings zu erheblichen Laufzeiten kommen.
Diese Routine wird ausgelöst von FRE(0) oder falls der Speicherplatz nicht mehr ausreicht.

Die Darstellung von Fliesskomma-Zahlen

In der Zeropage gibt es insgesamt sechs Fliesskomma-Register.
Dies sind die 'Floating Point Accus' FAC1 - FAC5, sowie ein Hilfsregister bei 62-65. Die Hauptrolle bei den Arithmetik-Operationen spielen jedoch FAC1 und FAC2. Ihr Format unterscheidet sich geringfügig vom allgemeinen Fliesskomma-Format, welches sich wie folgt zusammensetzt:
Das erste Byte stellt den Exponenten zur Basis 2 dar. Er bestimmt den Stellenwert der einzelnen Mantissen-Bits. Beträgt er 80, so hat das höchste Bit die Wertigkeit 0.5; bei 7F bzw. 81 beträgt diese entsprechend 0.25 bzw. 1.
Es folgen vier Mantissenbytes (M1 - M4), die Mantisse umfasst also 32 Bit.
Da der Stellenwert bereits durch den Exponenten bestimmt ist, muß das höchstwertigste Bit immer gesetzt sein. Da es somit keine Information trägt, dient es als Vorzeichenbit (minus = 1, plus = 0). Um die Zahl Null exakt darstellen zu können, wird der Exponent in diesem Fall nullgesetzt.

Beispiele:	83 40 00 00 00	bedeutet 6.0
	83 C1 00 00 00	bedeutet -6.03125

Bei FAC1 und FAC2 wird nun das Vorzeichen nicht ins erste Mantissenbit gesetzt, sondern in Bit7 eines gesonderten Vorzeichenbytes. Die obigen Beispiele ergeben dann

	00 83 C0 00 00 00	für 6.0
und	80 83 C1 00 00 00	für -6.03125

wobei das erste Byte jeweils das Vorzeichenbyte darstellt
Ausserdem besitzt FAC1 ein 5. Mantissenbyte als Rundungsstelle, um Rundungsfehler zu reduzieren.

Soll FAC1 ins Integerformat gebracht werden, so wird die Mantisse so weit verschoben, bis das letzte Bit den Stellenwert 1 hat. In obigen Beispielen ergibt sich

	A0 00 00 00 06	für 6.0
und	A0 FF FF FF F9	für -7.0

da negative Integerzahlen Vorzeichenrichtig dargestellt werden.

Strukturierung des Programmtextes

Der Anfang des Programmtextes wird durch den Zeiger 67,68 angegeben, die erste Adresse hinter dem Programmtext durch den Zeiger AF,B0.
Im Normalfall beginnt der Programmtext bei der Adresse $0801.
Innerhalb dieses Bereiches stehen die Programmzeilen in der Reihenfolge ihrer Zeilennummern angeordnet. Jede Programmzeile wird dabei wie folgt dargestellt:

- Linkadresse; dies ist die RAM-Adresse der folgenden Programmzeile.
- Zeilennummer; sie wird als 2-Byte-Integer dargestellt. Die kleinste erlaubte Zeilennummer ist 0, die größte 63999.
- Text der Programmzeile; bei der Programmzeilenkodierung wurden sämtliche erkannten Keywords durch die zugehörigen Tokens ersetzt.
- 00 kennzeichnet das Ende einer Programmzeile.

Das Ende des Programmtextes wird im Text selbst durch zwei Bytes mit 00 gekennzeichnet. Der Programtext endet also mit insgesamt dreimal 00.
Wird der Befehl NEW ausgeführt, so wird keineswegs der gesamte Speicherbereich gelöscht. Es werden lediglich an den Programmanfang zwei Nullbytes eingetragen, als Marke für das Programmtext-Ende, und die Zeiger in der Zeropage werden neu gesetzt. Durch die Korrektur dieser Bytes direkt im Speicher läßt sich der Text wieder retten.

Die CHRGET-Routine

Diese Routine holt ein Zeichen aus dem Programmtext, dem Eingabepuffer oder sonst einem Datenbereich, und klassifiziert dieses bereits.
Die Kaltstart-Routine kopiert die CHRGET-Routine von $F10B ff in die Zeropage ab $00B1. Der CHRGET-Zeiger B8,B9 stellt dort genau die Adresse eines absolut adressierten LDA-Befehls dar.
Es existieren zwei Einsprungstellen, $00B1 und $00B7. Die erstere erhöht den CHRGET-Zeiger um eine Position, um das nächste Zeichen zu holen. Beim zweiten Einsprung wird genau das aktuelle Zeichen geholt.
Falls dieses Zeichen ein Leerzeichen ist, wird es ignoriert, d.h. der CHRGET-Zeiger wird erhöht, und gleich das nächste Zeichen geholt.
Ansonsten kehrt die Routine mit dem Zeichen im Accu zurück, wobei die Statusflags C und Z wie folgt gesetzt sind:

- C=0, falls das Zeichen eine Ziffer ist;
  C=1, falls das Zeichen keine Ziffer ist;
- Z=0, falls das Zeichen weder 00 noch 3A ist;
  Z=1, falls das Zeichen 00 oder 3A ist;
  Sowohl 00 als auch 3A dienen als Trennzeichen in Basictexten.
  00 markiert das Ende einer Programmzeile oder Eingabezeile;
  3A ist der Ascii-Kode von des Zeichens ":", welches als Trennzeichen zwischen zwei Anweisungen einer Befehlszeile dient.

Durch prüfen dieser Flags wird die Analyse von Befehlszeilen erheblich vereinfacht.
Beim Aufruf einer Applesoft-Routine ($D828 ff) wird als letzte Operation vor dem Sprung zur Routine noch das nächste Zeichen geholt. Die einzelnen Routinen setzen deswegen vorraus, daß sich im Accu das nächste Zeichen befindet, und die beschriebenen Flags entsprechend gesetzt sind.

Die Routine zur Auswertung von Ausdrücken

Diese Routine ($DD7B ff) ist eines der wichtigsten Hilfsmittel des Interpreters, und zugleich eine der am schwersten verständlichen. Sie analysiert den Ausdruck, auf den der CHRGET-Zeiger zeigt.
Zunächst holt die Routine ab $DE60 einen Operanden. Folgt auf diesen ein Trennzeichen, wird die Routine sofort abgeschlossen (über $DE35). Folgt ein Vergleichszeichen, so wird die Vergleichsoperation bestimmt und das Vergleichsflag entsprechend gesetzt. Je nach dem Prioritätsflag auf dem Stack werden Daten dieser Operation (Routinen-Adresse, Operand, Vergleichsflag und Prioritätsflag) auf den Stack geschoben, oder aber, falls die offene Operation auf dem Stack den Vorrang hatte, diese mit dem neuen Operanden vollzogen.
Folgt einem String-Operanden ein "+", so wird eine Stringverknüpfung durchgeführt.
Handelt es sich jedoch um eine arithmetische Operation, so geschieht ähnliches, wie vorher bei der Vergleichsoperation beschrieben.
Dies soll nun durch eine kleines Beispiel erläutert werden.
Der Ausdruck sei 3 + 5 * 2, es geschieht nun folgendes:
- $DE60 bringt die 3 nach FAC1, der Operator "+" wird geholt;
- Prioritätsvergleich: Stackpriorität = 0, Priorität von "+" = 79
- $DDFD bringt die Sprungadresse für die Additionsroutine, den Operanden, und (ab $DD86) das Vergleichsflag (=0) und das Prioritätsflag (= 79) auf den Stack. Die Rücksprungadresse nach DDDA bleibt auf dem Stack.
- $DE60 bringt nun 5 nach FAC1, und der Operator "*" wird geholt;
- Prioritätsvergleich: Stack = 79, "*" = 7B, Multiplikation hat Vorrang;
- $DDFD bringt die Daten dieser Operation ebenfalls auf den Stack;
- $DE60 bringt 2 nach FAC1, das nächste Zeichen ist ein Trennzeichen;
- $DE35 bringt 5 nach FAC2, über RTS Sprung zur Multiplikationsroutine;
- nach deren Abschluss zurück nach $DDDA, dort Prioritätsvergleich mit der nächsten Stackoperation. Trennzeichen hat Priorität FF.
- $DE43 bringt 3 nach FAC2 (in FAC1 steht 10), über RTS zur Addition;
- nach deren Abschluss wieder zurück nach $DDDA. Routine abschliessen, da keine Operation mehr offen steht.

Die Routine ab $DE60, die jeweils den Operanden holt, unterscheidet folgende Fälle:
- numerische Konstante, gegeben als String. Es erfolgt die Umwandlung ins Fliesskommaformat, die Konstante steht dann in FAC1.
- Variablenwert. Die angegebene Variable wird gesucht, und der Inhalt nach FAC1 gebracht. (Bei Stringvariablen wird der Zeiger auf den String-Deskriptor nach A0,A1 gebracht.)
- Stringkonstante, gegeben in Ascii. Der Stringdeskriptor wird bestimmt und auf den Deskriptorenstack geschoben; A0,A1 zeigt auf den Deskriptor.
- FN-Ausdruck. Sprung zur FN-Auswertung, Ergebnis steht in FAC1.
- Funktionswert, z.B. SQR(8). Sprung zur Auswertung von Funktionen.
- Klammerausdruck. Hier wird nun zur Auswertung des Ausdrucks zwischen den Klammern die Routine zur Ausdrucksauswertung ab $DD7B gerufen. Durch die auf dem Stack gemerkten Rücksprungadressen wird diese rekursive Vorgehensweise ermöglicht.
- Wird keiner der vorstehenden Fälle erkannt, ist der Ausdruck fehlerhaft, und SYNTAX ERROR wird ausgelöst.

Die Routine ab $DEB8 wird vielfach zur Syntax-Prüfung benutzt. Sie vergleicht das Zeichen, auf welches CHRGET zeigt, mit ")", "(", "," oder einem sonstigen Ascii-Zeichen, je nach Einsprung. Bei Übereinstimmung erfolgt der Rücksprung mit dem nächsten Zeichen im Accu, sonst SYNTAX ERROR.

GRAFIKROUTINEN

Das Zeichnen einer Linie

Die Anweisung HPLOT x1,y1 TO x2,y2 ermöglicht das Zeichnen von beliebigen geraden Linien durch die Angabe der Endpunkte. Da der Bildschirm in ein 280 x 192 -Punktraster aufgeteilt ist, müssen schräge Linien durch eine stufenartige Figur angenähert werden.
Dazu werden zunächst der horizontale und vertikale Abstand der Endpunkte bestimmt. Das Verhältnis dieser beiden Strecken legt die Steigung der Geraden fest. Die Summe dieser beiden Strecken ergibt die Anzahl der zu zeichnenden Punkte.
Es wird nun ein Saldo-Register eingerichtet. Jedesmal, wenn ein Punkt in horizontaler Richtung gesetzt wurde, wird die Y-Differenz der Endpunkte vom Saldo subtrahiert; jedesmal, wenn ein Punkt in vertikaler Richtung gesetzt wurde, wird die X-Differenz der Endpunkte zum Saldo addiert. Bei jedem Vorzeichenwechsel des Saldoregisters wechselt die Zugrichtung.
Wenn also z.B. die Y-Differenz nur die Hälfte der X-Differenz beträgt, sind doppelt so viele Züge in horizontaler Richtung notwendig wie in vertikaler Richtung, um das Saldoregister auszugleichen, und es ergibt sich genau die richtige Steigung der Linie. Es wird hierbei mit den Beträgen der X- bzw. Y-Differenz der Endpunkte gearbeitet. Die beiden Vorzeichen werden in ein Richtungs-Flag integriert, welches die Zugrichtung vollends festlegt, also ob links oder rechts bei horizontalem Zug und nach oben oder nach unten bei vertikalem Zug. Nachdem die zuvor bestimmte Gesamtzahl von Zügen erfolgt ist, wird die Routine beendet.

Das Zeichnen eines Shape

Shapes erlauben auf elegante Weise die Darstellung beliebiger Figuren in hochauflösender Grafik. Die Definition eines Shape besteht aus Anweisungen von jeweils 3 Bit, die die Zugrichtung für den nächsten Punkt angeben, und ob der Punkt gesetzt werden soll. Die Darstellung auf dem Bildschirm wird durch 2 Parameter beeinflußt, den SCALE-Wert und den ROT-Wert.
Der SCALE-Wert gibt an, wie oft jede Anweisung wiederholt werden soll, und erlaubt so die vergrößerte Darstellung eines Shape. Der ROT-Wert gibt an, um wieviel der Shape im Uhrzeigersinn gedreht dargestellt werden soll. ROT=16 bedeutet die Drehung um 90'. Der ROT-Wert mod 16 ergibt so den Winkel gegenüber der letzten Koordinaten-Achse.
Die Routinen von DRAW und XDRAW, die Shapes darstellen, konstruieren schräge Linien ähnlich wie zuvor bei HPLOT geschildert. Aus einer Tabelle werden der Sinus und der Cosinus des Drehwinkels gegenüber der letzten Koordinatenachse bestimmt. Durch Führung von zwei Saldoregistern wird nun, je nach Verhältnis des Sinuswerts zum Cosinuswert, die Häufigkeit von Zügen in Richtung der letzten Koordinatenachse und von solchen in Richtung der nächsten Koordinatenachse im Uhrzeigersinn festgelegt. Je größer der Sinuswert, desto häufiger geht der Zug in Richtung der nächsten Achse, und desto größer wird somit der Winkel innerhalb dieses Quadranten.
Der Quadrant selbst wird festgelegt durch die Richtungsangabe der Shape-Anweisung und die Anzahl voller rechter Winkel des ROT-Wertes.
Beispiel: eine Shapeanweisung nach unten ergibt bei einem ROT-Wert von 40 folgende dargestellte Richtung: ROT = 40 bedeutet 225', d.h. 180' + 45'. Sin(45') = Cos(45'), die Linie halbiert also den 1. Quadranten, und der SCALE-Wert bestimmt die Länge der Linie.
Der Collision Counter zählt die Anzahl der Punkte, die beim Zeichnen eines Shape auf schon gesetzte Punkte treffen, d.h. die Anzahl der "Treffer".

Sprungtabelle für Applesoft-Befehle
(jeweils Anfangsadresse -1)

			Token
D000-	6F D8 65 D7	END, FOR	80, 81
D004-	F8 DC 94 D9	NEXT, DATA	82, 83
D008-	B1 DB 30 F3	INPUT, DEL	84, 85
D00C-	D8 DF E1 DB	DIM, READ	86, 87
D010-	8F F3 98 F3	GR, TEXT	88, 89
D014-	E4 F1 DD F1	PR#, IN#	8A, 8B
D018-	D4 F1 24 F2	CALL, PLOT	8C, 8D
D01C-	31 F2 40 F2	HLIN, VLIN	8E, 8F
D020-	D7 F3 E1 F3	HGR2, HGR	90, 91
D024-	E8 F6 FD F6	HCOLOR, HPLOT	92, 93
D028-	68 F7 6E F7	DRAW, XDRAW	94, 95
D02C-	E6 F7 57 FC	HTAB, HOME	96, 97
D030-	20 F7 26 F7	ROT=, SCALE=	98, 99
D034-	74 F7 6C F2	SHLOAD, TRACE	9A, 9B
D038-	6E F2 72 F2	NOTRACE, NORMAL	9C, 9D
D03C-	76 F2 7F F2	INVERSE, FLASH	9E, 9F
D040-	4E F2 6A D9	COLOR=, POP	A0, A1
D044-	55 F2 85 F2	VTAB, HIMEM:	A2, A3
D048-	A5 F2 CA F2	LOMEM:, ONERR	A4, A5
D04C-	17 F3 BB F3	RESUME, RECALL	A6, A7
D050-	9E F3 61 F2	STORE, SPEED	A8, A9
D054-	45 DA 3D D9	LET, GOTO	AA, AB
D058-	11 D9 C8 D9	RUN, IF	AC, AD
D05C-	48 D8 F4 03	RESTORE, &	AE, AF
D060-	20 D9 6A D9	GOSUB, RETURN	B0, B1
D064-	DB D9 6D D8	REM, STOP	B2, B3
D068-	EB D9 83 E7	ON, WAIT	B4, B5
D06C-	C8 D8 AF D8	LOAD, SAVE	B6, B7
D070-	12 E3 7A E7	DEF, POKE	B8, B9
D074-	D4 DA 95 D8	PRINT, CONT	BA, BB
D078-	A4 D6 69 D6	LIST, CLEAR	BC, BD
D07C-	9F DB 48 D6	GET, NEW	BE, BF

64 x B

Sprungtabelle für Funktionen
(jeweils Anfangsadresse)

			Token
D080-	90 EB 23 EC	SGN, INT	D2, D3
D084-	AF EB 0A 00	ABS, USR	D4, D5
D088-	DE E2 12 D4	FRE, SCRN(	D6, D7
D08C-	CD DF FF E2	PDL, POS	D8, D9
D090-	8D EE AE EF	SQR, RND	DA, DB
D094-	41 E9 09 EF	LOG, EXP	DC, DD
D098-	EA EF F1 EF	COS, SIN	DE, DF
D09C-	3A F0 9E F0	TAN, ATN	E0, E1
D0A0-	64 E7 D6 E6	PEEK, LEN	E2, E3
D0A4-	C5 E3 07 E7	STR$, VAL	E4, E5
D0A8-	E5 E6 46 E6	ASC, CHR$	E6, E7
D0AC-	5A E6 86 E6	LEFT$, RIGHT$	E8, E9
D0B0-	91 E6	MID$	EA

26 x 3

Sprungtabelle für Operationen
jeweils Prioritätswert und Routinenadresse -1

D0B2	79 C0 E7	Addition
D0B5	79 A9 E7	Subtraktion
D0B8	7B 81 E9	Multiplikation
D0BB	7B 68 EA	Division
D0BE	7D 96 EE	Potenzierung
D0C1	50 54 DF	logisch AND
D0C4	46 4E DF	logisch OR
D0C7	7F CF EE	Vorzeichenwechsel
D0CA	7F 97 DE	logisch NOT
D0CD	64 64 DF	Vergleichsoperation

Tabelle der Keywords in Ascii
(beim jeweils letzten Zeichen ist Bit7 gesetzt)

			Token
D0D0	45 4E C4	END	80
D0D3	46 4F D2	FOR	81
D0D6	4E 45 58 D4	NEXT	82
D0DA	44 41 54 C1	DATA	83
D0DE	49 4E 50 55 D4	INPUT	84
D0E3	44 45 CC	DEL	85
D0E6	44 49 CD	DIM	86
D0E9	52 45 41 C4	READ	87
D0ED	47 D2	GR	88
D0EF	54 45 58 D4	TEXT	89
D0F3	50 52 A3	PR#	8A
D0F6	49 4E A3	IN#	8B
D0F9	43 41 4C CC	CALL	8C
D0FD	50 4C 4F D4	PLOT	8D
D101	48 4C 49 CE	HLIN	8E
D105	56 4C 49 CE	VLIN	8F
D109	48 47 52 B2	HGR2	90
D10D	48 47 D2	HGR	91
D110	48 43 4F 4C 4F 52 BD	HCOLOR=	92
D117	48 50 4C 4F D4	HPLOT	93
D11C	44 52 41 D7	DRAW	94
D120	58 44 52 41 D7	XDRAW	95
D125	48 54 41 C2	HTAB	96
D129	48 4F 4D C5	HOME	97
D12D	52 4F 54 BD	ROT=	98
D131	53 43 41 4C 45 BD	SCALE=	99
D137	53 48 4C 4F 41 C4	SHLOAD	9A
D13D	54 52 41 43 C5	TRACE	9B
D142	4E 4F 54 52 41 43 C5	NOTRACE	9C
D149	4E 4F 52 4D 41 CC	NORMAL	9D
D14F	49 4E 56 45 52 53 C5	INVERSE	9E
D156	46 4C 41 53 C8	FLASH	9F
D15B	43 4F 4C 4F 52 BD	COLOR=	A0
D161	50 4F D0	POP	A1
D164	56 54 41 C2	VTAB	A2
D168	48 49 4D 45 4D BA	HIMEM:	A3
D16E	4C 4F 4D 45 4D BA	LOMEM:	A4
D174	4F 4E 45 52 D2	ONERR	A5
D179	52 45 53 55 4D C5	RESUME	A6
D17F	52 45 43 41 4C CC	RECALL	A7
D185	53 54 4F 52 C5	STORE	A8
D18A	53 50 45 45 44 BD	SPEED=	A9

D190	4C 45 D4	LET	AA
D193	47 4F 54 CF	GOTO	AB
D197	52 55 CE	RUN	AC
D19A	49 C6	IF	AD
D19C	52 45 53 54 4F 52 C5	RESTORE	AE
D1A3	A6	&	AF
D1A4	47 4F 53 55 C2	GOSUB	B0
D1A9	52 45 54 55 52 CE	RETURN	B1
D1AF	52 45 CD	REM	B2
D1B2	53 54 4F D0	STOP	B3
D1B6	4F CE	ON	B4
D1B8	57 41 49 D4	WAIT	B5
D1BC	4C 4F 41 C4	LOAD	B6
D1C0	53 41 56 C5	SAVE	B7
D1C4	44 45 C6	DEF	B8
D1C7	50 4F 4B C5	POKE	B9
D1CB	50 52 49 4E D4	PRINT	BA
D1D0	43 4F 4E D4	CONT	BB
D1D4	4C 49 53 D4	LIST	BC
D1D8	43 4C 45 41 D2	CLEAR	BD
D1DD	47 45 D4	GET	BE
D1E0	4E 45 D7	NEW	BF
D1E3	54 41 42 A8	TAB(	C0
D1E7	54 CF	TO	C1
D1E9	46 CE	FN	C2
D1EB	53 50 43 A8	SPC(	C3
D1EF	54 48 45 CE	THEN	C4
D1F3	41 D4	AT	C5
D1F5	4E 4F D4	NOT	C6
D1F8	53 54 45 D0	STEP	C7
D1FC	AB	+	C8
D1FD	AD	-	C9
D1FE	AA	*	CA
D1FF	AF	/	CB
D200	DE	^	CC
D201	41 4E C4	AND	CD
D204	4F D2	OR	CE
D206	BE	>	CF
D207	BD	=	D0
D208	BC	<	D1
D209	53 47 CE	SGN	D2
D20C	49 4E D4	INT	D3
D20F	41 42 D3	ABS	D4
D212	55 53 D2	USR	D5
D215	46 52 C5	FRE	D6
D218	53 43 52 4E A8	SCRN(	D7
D21D	50 44 CC	PDL	D8
D220	50 4F D3	POS	D9
D223	53 51 D2	SQR	DA
D226	52 4E C4	RND	DB
D229	4C 4F C7	LOG	DC
D22C	45 58 D0	EXP	DD
D22F	43 4F D3	COS	DE
D232	53 49 CE	SIN	DF
D235	54 41 CE	TAN	E0
D238	41 54 CE	ATN	E1
D23B	50 45 45 CB	PEEK	E2
D23F	4C 45 CE	LEN	E3
D242	53 54 52 A4	STR$	E4
D246	56 41 CC	VAL	E5
D249	41 53 C3	ASC	E6

D24C	43 48 52 A4	CHR$	E7
D250	4C 45 46 54 A4	LEFT$	E8
D255	52 49 47 48 54 A4	RIGHT$	E9
D25B	4D 49 44 A4	MID$	EA
D25F	00	als Endmarke	

Tabelle der Fehlermeldungen in Ascii mit den zugehörigen Fehlerkodes

D260	4E 45 58 54 20 57 49 54 48 4F 55 54	NEXT WITHOUT	
	20 46 4F D2	FOR	00
D270	53 59 4E 54 41 D8	SYNTAX	10
D276	52 45 54 55 52 4E 20	RETURN	
	57 49 54 48 4F 55 54 20 47 4F 53 55 C2	WITHOUT GOSUB	16
D28A	4F 55 54 20 4F 46 20 44 41 54 C1	OUT OF DATA	2A
D295	49 4C 4C 45 47 41 4C 20	ILLEGAL	
	51 55 41 4E 54 49 54 D9	QUANTITY	35
D2A5	4F 56 45 52 46 4C 4F D7	OVERFLOW	45
D2AD	4F 55 54 20 4F 46 20	OUT OF	
	4D 45 4D 4F 52 D9	MEMORY	4D
D2BA	55 4E 44 45 46 27 44 20	UNDEF'D	
	53 54 41 54 45 4D 45 4E D4	STATEMENT	5A
D2CB	42 41 44 20 53 55 42 53 43 52 49 50 D4	BAD SUBSCRIPT	6B
D2D8	52 45 44 49 4D 27 44 20 41 52 52 41 D9	REDIM'D ARRAY	78
D2E5	44 49 56 49 53 49 4F 4E 20	DIVISION	
	42 59 20 5A 45 52 CF	BY ZERO	85
D2F5	49 4C 4C 45 47 41 4C 20 44 49 52 45 43 D4	ILLEGAL DIRECT	95
D303	54 59 50 45 20 4D 49 53 4D 41 54 43 C8	TYPE MISMATCH	A3
D310	53 54 52 49 4E 47 20 54 4F 4F 20	STRING TOO	
	4C 4F 4E C7	LONG	B0
D31F	46 4F 52 4D 55 4C 41 20 54 4F 4F 20	FORMULA TOO	
	43 4F 4D 50 4C 45 D8	COMPLEX	BF
D332	43 41 4E 27 54 20	CAN'T	
	43 4F 4E 54 49 4E 55 C5	CONTINUE	D2
D340	55 4E 44 45 46 27 44 20	UNDEF'D	
	46 55 4E 43 54 49 4F CE	FUNCTION	E0

Gemeinsame Texte

D350	20 45 52 52 4F 52 07 00	" ERROR" + Glocke
D358	20 49 4E 20 00	" IN "
D35D	0D 42 52 45 41 4B 07 00	CR + "BREAK" + Glocke

Kommentiertes Listing der ROM-Routinen von $D365 - $F7FF

Allgemeine Bemerkungen

Im Listing wurden zusammenhängende Routinen gegeneinander abgegrenzt, um eine übersichtlichere Darstellung zu erhalten.
Den einzelnen Routinen ist jeweils eine Überschrift beigefügt, die bei Basic-Befehlen das entsprechende Keyword enthält, und bei allen übrigen Routinen (z.B. Unterprogrammen) eine stichwortartige Funktionsbeschreibung.
Lange, und unübersichtliche Routinen wurden durch gelegentliche Zwischenüberschriften übersichtlicher dargestellt.
Wo dies sinnvoll oder notwendig erschien, wurden zusätzlich noch die Schnittstellen spezifiziert, d.h. die Parameter, die die Routine beim Einsprung erwartet, sowie jene, die nach Abschluss der Routine bereitstehen. Dabei kennzeichnet ein vorgestelltes "I:" die Einsprungparameter, und ein vorgestelltes "O:" die Rücksprungparameter.
Der Einfachkeit halber wurden jedoch folgende Parameter nicht mit aufgeführt, da sie von allgemeiner Natur sind:

- Der CHRGET-Zeiger zeigt bei Applesoft-Routinen auf das erste Zeichen hinter dem zugehörigen Token, der Accu enthält dieses Zeichen, und die Statusflags C und Z sind entsprechend diesem Zeichen gesetzt. (vgl. die Bemerkung zur CHRGET-Routine)
  Das Y-Register enthält beim Einsprung (Token - 80) x 2 (vgl. $D828 ff)
- Nach Abschluss von Applesoft-Routinen zeigt der CHRGET-Zeiger auf das Trennzeichen hinter dem Basic-Statement.
- Applesoft-Funktionen (Token D2 - E7) erwarten folgende Einsprungparameter: Das Argument befindet sich im FAC1, der CHRGET-Zeiger zeigt bereits auf das nächste Trennzeichen. Bei Stringfunktionen steht der Zeiger auf den Stringdeskriptor in A0,A1.
  Beim Rücksprung befinden sich numerische Ergebnisse in FAC1, bei Strings steht der Deskriptor im Deskriptoren-Stack, und A0,A1 zeigt darauf.

Zahlen im Kommentar sind in der Regel in dezimaler Notation angegeben, um die Lesbarkeit zu erhöhen. Unter #$36(hex) z.B. kann man sich weniger vorstellen wie unter 54(dez). In seltenen Fällen (besonders bei FF) wurde die hexadezimale (eigentlich: sedezimale) Notation verwendet, wo dies zum Verständnis beitrug, und durch ein vorgesetztes "$" gekennzeichnet. Zeropageadressen jedoch sind immer in hexadezimaler Notation angegeben.

Es wurde versucht, das Assembler-Listing nicht zu paraphrasieren, sondern wirklich zu kommentieren. Es wurde deswegen nicht jeder trivialen Zeile ein Kommentar beigefügt. Insbesondere bei dem häufigen Umgang mit 2-Byte-Zeigern wurde der gesamte Vorgang (z.B. Zeiger eintragen oder erhöhen) meist in einem einzigen Kommentar am Anfang zusammengefasst, um die Übersichtlichkeit zu erhöhen. Auch bei kurzen Schleifen wurde häufig etwas pauschaler kommentiert.

Bei bedingten Sprüngen wurde beim Sprungbefehl die Bedingung bzw. anschauliche Begründung für den Sprung angeführt.

Übersicht über die ROM-Routinen, nach Adressen sortiert

D365	Suche FOR/NEXT-Parameter im Stack, Unterprogramm
D39A	Speicherblock verschieben, Unterprogramm
D3D6	Platz auf Stack prüfen, Unterprogramm
D3E3	freien Speicherplatz prüfen, Unterprogramm
D412	Fehlermeldungen behandeln
D43C	Warmstart
D45C	neue Programmzeile übernehmen
D52C	Eingabe einer Befehlszeile, Unterprogramm
D553	Eingabe eines einzelnen Zeichens, Unterprogramm
D559	Applesoft Keywords kodieren, Unterprogramm
D61A	Programmzeile suchen, Unterprogramm
D649	NEW, Routine
D66A	CLEAR, Routine
D683	Stack initialisieren, Unterprogramm
D697	CHRGET auf Programmstart -1 setzen, Unterprogramm
D6A5	LIST, Routine
D766	FOR, Routine
D7D2	Interpreter-Hauptschleife
D849	RESTORE, Routine
D86E	STOP, Routine
D870	END, Routine
D896	CONT, Routine
D8B0	SAVE, Routine
D8C9	LOAD, Routine
D8F0	setze Monitorregister für Längendaten, Unterprogramm
D901	setze Monitorregister für Programmtext
D912	RUN, Routine
D921	GOSUB, Routine
D93E	GOTO, Routine
D96B	RETURN/POP, Routine
D9A3	suche nächstes Trennzeichen, Unterprogramm
D9A5	suche Zeilenende, Unterprogramm
D9C9	IF, Routine
D9DC	REM, Routine
D9EC	ON, Routine
DA0C	hole Zeilennummer aus Programmtext, Unterprogramm
DA46	LET, Routine
DA7A	Deskriptor in Variable eintragen
DAD5	PRINT, Routine
DB3A	String drucken, Unterprogramm
DB57	einzelnes Zeichen drucken, Unterprogramm
DBA0	GET, Routine
DBB2	INPUT, Routine
DBE2	READ, Routine
DCF9	NEXT, Routine
DD6A	prüfe, ob Ausdruck numerisch
DD6C	prüfe, ob Stringausdruck
DD7B	Auswertung eines beliebigen Ausdrucks, Unterprogramm
DE60	hole Operand aus Programmtext, Unterprogramm
DE98	NOT, Operation
DED5	hole Variablenwert, Unterprogramm
DEF9	SCRN, Routine
DF0C	Auswertung von Funktionen
DF4F	OR, Operation
DF55	AND, Operation
DF65	Vergleich durchführen
DFCD	PDL, Routine
DFD9	DIM, Routine
DFE3	suche Variable, Unterprogramm
E07D	prüfe Zeichen auf Buchstabe, Unterprogramm

E2AD	Multiplikation, Unterprogramm
E2DE	FRE, Routine
E2F2	Integer (A,Y) --> FAC1
E2FF	POS, Routine
E313	DEF, Routine
E354	FN auswerten
E3C5	STR$, Funktion
E3E7	Stringdeskriptor bestimmen
E452	Platz für neuen String schaffen
E484	Garbage Collection
E597	Stringverkettung, Operation
E5D4	String kopieren, Unterprogramm
E5FD	unnütze Strings entfernen, Unterprogramm
E646	CHR$, Funktion
E65A	LEFT$, Funktion
E686	RIGHT$, Funktion
E691	MID$, Funktion
E6D6	LEN, Funktion
E6E5	ASC, Funktion
E6F5	hole 1-Byte-Integer aus Programmtext, Unterprogramm
E707	VAL, Funktion
E746	hole 2-Byte-Integer und 1-Byte Integer, Unterprogramm
E752	FAC1 --> Integer
E764	PEEK, Funktion
E77B	POKE, Routine
E784	WAIT, Routine
E7A0	FAC1 = FAC1 + 0.5, Unterprogramm
E7A7	FAC1 = (A,Y) - FAC1, Unterprogramm
E7AA	FAC1 = FAC2 - FAC1, Operation
E7BE	FAC1 = (A,Y) + FAC1, Unterprogramm
E7C6	FAC1 = FAC2 + FAC1, Operation
E941	LOG, Funktion
E97F	FAC1 = (A,Y) * FAC1, Unterprogramm
E987	FAC1 = FAC2 * FAC1, Operation
EA39	FAC1 = 10 * FAC1, Unterprogramm
EA55	FAC1 = FAC1 / 10, Unterprogramm
EA66	FAC1 = (A,Y) / FAC1, Unterprogramm
EA69	FAC1 = FAC2 / FAC1, Operation
EAF9	(A,Y) --> FAC1
EB2B	FAC1 --> (X,Y)
EB53	FAC2 --> FAC1
EB63	FAC1 --> FAC2, Unterprogramme
EB72	FAC1 runden, Unterprogramm
EB82	Vorzeichen von FAC1 testen, Unterprogramm
EB90	SGN, Funktion
EBAF	ABS, Funktion
EBB2	FAC1 mit (A,Y) vergleichen, Unterprogramm
EBF2	FAC1 --> Integer, Unterprogramm
EC23	INT, Funktion
EC4A	Zahlenstring --> FP-Konstante, Unterprogramm
ED34	FAC1 --> Zahlenstring, Unterprogramm
EE8D	SQR, Funktion
EE97	FAC1 = FAC2 ^ FAC1, Operation
EF09	EXP, Funktion
EF5C	ungerades Polynom auswerten, Unterprogramm
EF72	Polynom auswerten, Unterprogramm
EFAE	RND, Funktion
EFEA	COS, Funktion
EFF1	SIN, Funktion
F03A	TAN, Funktion
F09E	ATN, Funktion

F10B	Kopie der CHRGET-Routine
F128	Kaltstart
F1D5	CALL, Routine
F1DE	IN#, Routine
F1E5	PR#, Routine
F1EC	hole Plot-Parameter für LORES-Grafik, Unterprogramm
F209	hole Parameter für HLIN/VLIN, Unterprogramm
F225	PLOT, Routine
F232	HLIN, Routine
F241	VLIN, Routine
F24F	COLOR, Routine
F256	VTAB, Routine
F262	SPEED, Routine
F26D	TRACE, Routine
F26F	NOTRACE, Routine
F273	NORMAL, Routine
F277	INVERSE, Routine
F280	FLASH, Routine
F286	HIMEM:, Routine
F2A6	LOMEM:, Routine
F2CB	ONERR, Routine
F2E9	GOTO nach ONERR, Routine
F318	RESUME, Routine
F331	DEL, Routine
F390	GR, Routine
F399	TEXT, Routine
F39F	STORE, Routine
F3BC	RECALL, Routine
F3D8	HGR2, Routine
F3E3	HGR, Routine
F411	XY-Koordinaten --> RAM-Adresse, Unterprogramm
F457	Grafikpunkt setzen, Unterprogramm
F465	RAM-Adresse für horizontalen Nachbarpunkt bestimmen, Unterprogramm
F49C	Shapeanweisung durchführen für XDRAW, Unterprogramm
F4B3	Shapeanweisung durchführen für DRAW, Unterprogramm
F4D3	RAM-Adresse für vertikalen Nachbarpunkt bestimmen, Unterprogramm
F53A	Linie zeichnen in HIRES-Grafik, Unterprogramm
F5CB	RAM-Adresse --> XY-Koordinaten, Unterprogramm
F605	DRAW, Unterprogramm
F661	XDRAW, Unterprogramm
F6B9	hole Parameter für HPLOT, DRAW, XDRAW, Unterprogramm
F6E9	HCOLOR, Routine
F6FE	HPLOT, Routine
F721	ROT=, Routine
F727	SCALE=, Routine
F72D	hole Parameter für DRAW/XDRAW, Unterprogramm
F769	DRAW, Routine
F76F	XDRAW, Routine
F775	SHLOAD, Routine
F7BC	Monitorregister für STORE/RECALL setzen, Unterprogramm
F7D9	suche Feld für STORE/RECALL, Unterprogramm
F7E7	HTAB, Routine

Suche FOR/NEXT-Parameter im Stack
I: 85,86 NEXT-Variable falls angegeben, sonst 86=0
O: gefunden -> Z=1, X=S, A=Var.NameLB, sonst Z=0

```
D365-   TSX                 zwei Return-Adressen überspringen
D366-   INX
D367-   INX
D368-   INX
D369-   INX
D36A-   LDA   $0101,X
D36D-   CMP   #$81          Kode für FOR/NEXT-Parameter?
D36F-   BNE   $D392         nein, zurück mit Byte im Accu
D371-   LDA   $86           Laufvariable HByte
D373-   BNE   $D37F         nicht null, d.h. NEXT-Variable angegeben
D375-   LDA   $0102,X       FOR-Variable der letzten offenen Schleife übernehmen
D378-   STA   $85
D37A-   LDA   $0103,X
D37D-   STA   $86
D37F-   CMP   $0103,X       Übereinstimmung mit gemerkter Variablen?
D382-   BNE   $D38B         nein, weitersuchen
D384-   LDA   $85           Laufvariable LByte
D386-   CMP   $0102,X       Übereinstimmung?
D389-   BEQ   $D392         ja, aktuelle Schleife gefunden --> Return
D38B-   TXA                 sonst 18 Positionen überspringen und weitersuchen
D38C-   CLC
D38D-   ADC   #$12
D38F-   TAX
D390-   BNE   $D36A
D392-   RTS
---------------------
```

Platz für Speicherblock-Verschiebung?
I: A,Y = neues erwünschtes Blockende

```
D393-   JSR   $D3E3         prüft ob genügend freier Speicherplatz
D396-   STA   $6D           neues Ende der Feldvariablen
D398-   STY   $6E
```

Speicherblock verschieben
I: 9B,9C alter Anfang; 96,97 altes Ende+1;
   94,95 neues Ende+1 (94,95 > 96,97)
O: 96,97+1 = 9B,9C = alter Anfang; 94,95+1 neuer Anf.
   X = 0; Y = 0; A = letztes Byte

```
D39A-   SEC
D39B-   LDA   $96           altes Ende+1,LByte
D39D-   SBC   $9B           alter Blockanfang,LByte
D39F-   STA   $5E           Blocklänge modulo 256
D3A1-   TAY
D3A2-   LDA   $97           altes Ende+1,HByte
D3A4-   SBC   $9C           alter Anfang,HByte
D3A6-   TAX                 Anzahl voller Blöcke zu je 256 Byte
D3A7-   INX
D3A8-   TYA                 Y = 0?
D3A9-   BEQ   $D3CE         ja, kein Restblock  -> volle Blöcke verschieben
D3AB-   LDA   $96           altes Ende+1,LByte
D3AD-   SEC
D3AE-   SBC   $5E           Länge des Restblocks
D3B0-   STA   $96           Anfang Restblock,LByte
D3B2-   BCS   $D3B7
D3B4-   DEC   $97           Anfang Restblock,HByte
D3B6-   SEC
D3B7-   LDA   $94           neues Ende+1,LByte
D3B9-   SBC   $5E           Länge des Restblocks abziehen,
```

```
D3BB-   STA   $94        wird durch Y-Indizierung ersetzt  ( 5E = Y )
D3BD-   BCS   $D3C7
D3BF-   DEC   $95
D3C1-   BCC   $D3C7
D3C3-   LDA   ($96),Y    Verschiebeschleife für Restblock
D3C5-   STA   ($94),Y
D3C7-   DEY
D3C8-   BNE   $D3C3
D3CA-   LDA   ($96),Y    Verschiebeschleife für volle Blöcke
D3CC-   STA   ($94),Y
D3CE-   DEC   $97
D3D0-   DEC   $95
D3D2-   DEX
D3D3-   BNE   $D3C7      weiteren vollen Block verschieben
D3D5-   RTS
--------------------
```

Prüfung, ob Platz auf Stack ausreicht
I: Accu x 2 = benötigter Platz auf Stack
O: falls ok, X = S; C = 1

```
D3D6-   ASL              x 2
D3D7-   ADC   #$36
D3D9-   BCS   $D410      weniger als 54 Bytes in Reserve
D3DB-   STA   $5E
D3DD-   TSX              freie Bytes auf Stack
D3DE-   CPX   $5E
D3E0-   BCC   $D410      bleiben weniger als 54 -> OUT OF MEMORY ERROR
D3E2-   RTS
--------------------
```

Prüfung, ob Platz im Speicher ausreicht
I: A,Y = gewünschtes Ende; 6F,70 = max. Ende
O: falls ok, A,Y = gewünschtes Ende; C = 0

```
D3E3-   CPY   $70        prüfe HByte
D3E5-   BCC   $D40F      ok, fertig
D3E7-   BNE   $D3ED
D3E9-   CMP   $6F        prüfe LByte
D3EB-   BCC   $D40F      ok, fertig
D3ED-   PHA              rette A,Y auf den Stack
D3EE-   LDX   #$09       Zähler für Schleife
D3F0-   TYA
D3F1-   PHA
D3F2-   LDA   $93,X      rette FAC3 und FAC4 auf den Stack
D3F4-   DEX
D3F5-   BPL   $D3F1
D3F7-   JSR   $E484      Garbage Collection
D3FA-   LDX   #$F7       Zähler für Schleife
D3FC-   PLA              hole FAC3 und FAC4 wieder vom Stack
D3FD-   STA   $9D,X
D3FF-   INX
D400-   BMI   $D3FC
D402-   PLA              hole A,Y wieder vom Stack
D403-   TAY
D404-   PLA
D405-   CPY   $70        prüfe erneut, ob A,Y < 6F,70
D407-   BCC   $D40F      ok
D409-   BNE   $D410      -> OUT OF MEMORY ERROR
D40B-   CMP   $6F
D40D-   BCS   $D410
D40F-   RTS
--------------------
```

```
D410-   LDX   #$4D        Kode für OUT OF MEMORY ERROR

Behandlung von Fehlermeldungen
I: X = Fehlerkode
D412-   BIT   $D8         teste ON ERR -Flag
D414-   BPL   $D419       nicht gesetzt, Fehlermeldung drucken
D416-   JMP   $F2E9       zur Fehlerbehandlung
D419-   JSR   $DAFB       CR drucken
D41C-   JSR   $DB5A       ?  drucken
D41F-   LDA   $D260,X     Fehlermeldung drucken, X als Zeiger in Tabelle
D422-   PHA               Zeichen retten
D423-   JSR   $DB5C       Zeichen drucken
D426-   INX
D427-   PLA               hole letztes Zeichen wieder
D428-   BPL   $D41F       Bit7 = 0, d.h. es kommen noch weitere Zeichen
D42A-   JSR   $D683       Stack und Deskriptorenstack initialisieren
D42D-   LDA   #$50        A,Y = Zeiger auf ERROR, Bell
D42F-   LDY   #$D3
D431-   JSR   $DB3A       drucken
D434-   LDY   $76         = $FF, falls Direktmodus
D436-   INY
D437-   BEQ   $D43C
D439-   JSR   $ED19       IN (Zeilennummer) drucken

Warmstart
D43C-   JSR   $DAFB       CR drucken
D43F-   LDX   #$DD        Ascii für Promptzeichen
D441-   JSR   $D52E       Eingabezeile empfangen  (X,Y = $1FF; A = 0)
D444-   STX   $B8         CHRGET-Zeiger auf Tastatur-Puffer
D446-   STY   $B9
D448-   LSR   $D8         lösche ON ERR -Flag  (Bit7 = 0)
D44A-   JSR   $00B1       hole erstes Zeichen
D44D-   TAX               Z aktualisieren
D44E-   BEQ   $D43C       kein Eingabezeichen  --> Warmstart
D450-   LDX   #$FF
D452-   STX   $76         setzte Direktmodus
D454-   BCC   $D45C       C=0, d.h. Ziffer (vgl $00B1) --> Programmzeile
D456-   JSR   $D559       Basic Keywords kodieren
D459-   JMP   $D805       Befehlszeile ausführen
-----------------------

Neue Programmzeile übernehmen
I: A = 1. Ziffer der Zeilennummer
D45C-   LDX   $AF         Programmtextende +1
D45E-   STX   $69         = Variablenanfang, damit Variablen direkt hinter
D460-   LDX   $B0                            dem Programmtext
D462-   STX   $6A
D464-   JSR   $DA0C       hole Zeilennummer aus Text --> 50,51
D467-   JSR   $D559       Applesoft Keywords kodieren
D46A-   STY   $0F         Länge der neuen Zeile
D46C-   JSR   $D61A       Zeilennummer schon benutzt?  C=1 falls ja
D46F-   BCC   $D4B5       nein, neue Zeile einfügen

alte Zeile löschen
D471-   LDY   #$01
D473-   LDA   ($9B),Y     AdrH der folgenden Programmzeile
D475-   STA   $5F         als Basiszeiger auf Quellbereich, HByte
D477-   LDA   $69         Ende des Programmtextes, LByte
D479-   STA   $5E         als Basiszeiger auf Quellbereich, LByte
D47B-   LDA   $9C         AdrH der zu löschenden Zeile
D47D-   STA   $61         als Basiszeiger auf Zielbereich, HByte
```

```
D47F-   LDA   $9B        AdrL der zu löschenden Zeile
D481-   DEY
D482-   SBC   ($9B),Y    minus AdrL der folgenden Zeile  (C war 1)
D484-   CLC              ergibt negative Zeilenlänge
D485-   ADC   $69        addiert zum bisherigen Programmtextende, LByte
D487-   STA   $69        neues Ende des Programmtextes, LByte
D489-   STA   $60        als Basiszeiger auf Zielbereich, LByte
D48B-   LDA   $6A
D48D-   ADC   #$FF       HByte korrigieren, falls nötig  (d.h. falls C=0)
D48F-   STA   $6A
D491-   SBC   $9C        minus AdrH der alten Zeile
D493-   TAX              ergibt Anzahl zu verschiebender 256-Byte-Blöcke
D494-   SEC
D495-   LDA   $9B        AdrL der alten Zeile
D497-   SBC   $69        minus neues Ende des Programmtextes
D499-   TAY              ergibt Index auf erstes zu verschiebendes Byte
D49A-   BCS   $D49F
D49C-   INX              Blockzahl korrigieren
D49D-   DEC   $61        Basiszeiger auf Zielbereich, HByte
D49F-   CLC
D4A0-   ADC   $5E        Index plus Basiszeiger auf Quellbereich, LByte
D4A2-   BCC   $D4A7      Basiszeiger auf Quellbereich korrigieren falls nötig
D4A4-   DEC   $5F
D4A6-   CLC
D4A7-   LDA   ($5E),Y    Verschiebeschleife:  X zählt volle 256-Byte-Blocks
D4A9-   STA   ($60),Y                         Y Zeiger innerhalb eines Blocks
D4AB-   INY
D4AC-   BNE   $D4A7
D4AE-   INC   $5F        neuer 256-Byte-Block
D4B0-   INC   $61
D4B2-   DEX
D4B3-   BNE   $D4A7      weitere Blocks

Neue Programmzeile einfügen
D4B5-   LDA   $0200      erstes Zeichen
D4B8-   BEQ   $D4F2      Leerzeile, wird nicht eingefügt
D4BA-   LDA   $73        unteres Stringende := HIMEM  (damit Strings gelöscht)
D4BC-   LDY   $74
D4BE-   STA   $6F
D4C0-   STY   $70
D4C2-   LDA   $69        Ende des Programmtextes+1, LByte
D4C4-   STA   $96        altes Blockende+1, LByte  (für Verschieberoutine)
D4C6-   ADC   $0F        Länge der neuen Zeile
D4C8-   STA   $94        neues Blockende+1, LByte
D4CA-   LDY   $6A
D4CC-   STY   $97        altes Blockende+1, HByte
D4CE-   BCC   $D4D1
D4D0-   INY
D4D1-   STY   $95        neues Blockende+1, HByte
D4D3-   JSR   $D393      Block verschieben
D4D6-   LDA   $50        Zeilennummer
D4D8-   LDY   $51
D4DA-   STA   $01FE      vor Eingabepuffer setzen
D4DD-   STY   $01FF
D4E0-   LDA   $6D        neues Ende des Programmtextes +1
D4E2-   LDY   $6E
D4E4-   STA   $69        eintragen in Variablenanfangs-Vektor
D4E6-   STY   $6A
D4E8-   LDY   $0F        Zeilenlänge
D4EA-   LDA   $01FB,Y    neue Zeile einfügen
D4ED-   DEY
```

```
D4EE-   STA   ($9B),Y
D4F0-   BNE   $D4EA
D4F2-   JSR   $D665       CLEAR ausführen

Linkadressen neu berechnen
D4F5-   LDA   $67         Anfang des Programmtextes
D4F7-   LDY   $68
D4F9-   STA   $5E         auf Zeiger übertragen
D4FB-   STY   $5F
D4FD-   CLC
D4FE-   LDY   #$01
D500-   LDA   ($5E),Y     alte Linkadresse, HByte
D502-   BNE   $D50F       Programmende noch nicht erreicht
D504-   LDA   $69         Linkadressen komplett, Programmendevektor eintragen
D506-   STA   $AF
D508-   LDA   $6A
D50A-   STA   $B0
D50C-   JMP   $D43C       --> Warmstart
D50F-   LDY   #$04        Offset auf erstes Befehlszeichen
D511-   INY               suche Zeilenende (letztes Zeichen = 0)
D512-   LDA   ($5E),Y
D514-   BNE   $D511       weitersuchen
D516-   INY               Zeilenlänge
D517-   TYA
D518-   ADC   $5E         zu Zeiger addieren und als Linkadresse eintragen
D51A-   TAX               Linkadresse LByte merken
D51B-   LDY   #$00
D51D-   STA   ($5E),Y     eintragen
D51F-   LDA   $5F
D521-   ADC   #$00
D523-   INY
D524-   STA   ($5E),Y     HByte eintragen
D526-   STX   $5E         Zeiger aktualisieren
D528-   STA   $5F
D52A-   BCC   $D4FE       weiter mit nächster Zeile  (unbedingter Sprung)
-----------------------

Eingabe einer Zeile
D52C-   LDX   #$80
D52E-   STX   $33         Ascii-Kode für Prompt-Zeichen
D530-   JSR   $FD6A       Zeile empfangen über Monitorroutine GETLN
D533-   CPX   #$EF        X = Zeilenlänge, maximal 239 Zeichen
D535-   BCC   $D539
D537-   LDX   #$EF
D539-   LDA   #$00
D53B-   STA   $0200,X     0 als Endmarke anfügen
D53E-   TXA               Z-Flag aktualisieren
D53F-   BEQ   $D54C       Leerzeile, fertig
D541-   LDA   $01FF,X     in sämtlichen Zeichen Bit7 nullsetzen
D544-   AND   #$7F
D546-   STA   $01FF,X
D549-   DEX
D54A-   BNE   $D541       weitere Zeichen
D54C-   LDA   #$00
D54E-   LDX   #$FF        X,Y zeigt auf Eingabepuffer -1
D550-   LDY   #$01
D552-   RTS
-----------------------

Empfange ein einzelnes Zeichen
D553-   JSR   $FD0C       über Monitorroutine RDKEY
```

```
D556-   AND   #$7F       Bit7 nullsetzen
D558-   RTS
--------------------

Applesoft Keywords kodieren
D559-   LDX   $B8        Zeiger in Eingabepuffer
D55B-   DEX
D55C-   LDY   #$04       Zeiger vorbereiten (vgl. $D5B0)
D55E-   STY   $13        Bit6 nullsetzen  (DATA-Flag)
D560-   BIT   $D6        teste Autostart-Flag
D562-   BPL   $D56C      kein Autostart
D564-   PLA              entferne Rücksprungadresse vom Stack
D565-   PLA
D566-   JSR   $D665      CLEAR ausführen
D569-   JMP   $D7D2      Programm starten
D56C-   INX
D56D-   LDA   $0200,X    Zeichen aus Tastaturpuffer

D570-   BIT   $13        DATA-Flag
D572-   BVS   $D578      gesetzt, Leerzeichen nicht ignorieren
D574-   CMP   #$20       Ascii für Leerzeichen
D576-   BEQ   $D56C      ignorieren, neues Zeichen holen
D578-   STA   $0E        sonst merken
D57A-   CMP   #$22       Anführungszeichen?
D57C-   BEQ   $D5F2      ja, String übernehmen
D57E-   BVS   $D5CD      DATA-Statement, nicht kodieren
D580-   CMP   #$3F       Fragezeichen?
D582-   BNE   $D588      nein, weiter
D584-   LDA   #$BA       Token für PRINT
D586-   BNE   $D5CD      unbedingter Sprung
D588-   CMP   #$30
D58A-   BCC   $D590      keine Ziffer
D58C-   CMP   #$3C
D58E-   BCC   $D5CD      Ziffer, ":" oder ";" nicht kodieren
D590-   STY   $AD        Y retten
D592-   LDA   #$D0       9D,9E  Zeiger auf Befehlstabelle  (zunächst $CFD0)
D594-   STA   $9D
D596-   LDA   #$CF
D598-   STA   $9E
D59A-   LDY   #$00
D59C-   STY   $0F        Token-Zähler löschen
D59E-   DEY              Y=$FF, damit zeigt  (9D),Y auf  $D0CF
D59F-   STX   $B8        rette Zeiger für Eingabe-Puffer
D5A1-   DEX
D5A2-   INY              erhöhe Zeiger für Befehlstabelle
D5A3-   BNE   $D5A7
D5A5-   INC   $9E
D5A7-   INX
D5A8-   LDA   $0200,X    hole Zeichen
D5AB-   CMP   #$20       Leerzeichen?
D5AD-   BEQ   $D5A7      ja, ignorieren
D5AF-   SEC
D5B0-   SBC   ($9D),Y    mit Zeichen aus Befehlstabelle vergleichen
D5B2-   BEQ   $D5A2      stimmt überein, vergleiche nächstes Zeichen
D5B4-   CMP   #$80
D5B6-   BNE   $D5F9      Zeichen stimmt nicht, mit nächstem Befehl versuchen
D5B8-   ORA   $0F        Accu = Token, Bit7 = 1
D5BA-   CMP   #$C5       Token für AT?
D5BC-   BNE   $D5CB      nein
D5BE-   LDA   $0201,X    hole folgendes Zeichen
D5C1-   CMP   #$4E       Ascii für "N" ?
D5C3-   BEQ   $D5F9      ja, Interpretation als ATN
D5C5-   CMP   #$4F       Ascii für "O" ?
```

D5BE: JMP D5CB

```
D5C7-   BEQ   $D5F9       ja, Interpretation als  A TO
D5C9-   LDA   #$C5        sonst Interpretation als AT
D5CB-   LDY   $AD         hole gerettetes Y (Zeiger für kodierte Zeile)

Einsprung bei ?, :, ;, 0...9, oder DATA
D5CD-   INX               Zeiger erhöhen
D5CE-   INY
D5CF-   STA   $01FB,Y     Zeichen oder Token eintragen
D5D2-   LDA   $01FB,Y     Z-Flag aktualisieren
D5D5-   BEQ   $D610       Endmarke erreicht, Kodierung beendet
D5D7-   SEC
D5D8-   SBC   #$3A        Ascii für ":" ?
D5DA-   BEQ   $D5E0       ja, DATA-Flag löschen  (falls gesetzt)
D5DC-   CMP   #$49        $49 + $3A = $83, d.h. Token für DATA?
D5DE-   BNE   $D5E2
D5E0-   STA   $13         ja, DATA-Flag setzen  (d.h. Bit6 = 1)
D5E2-   SEC
D5E3-   SBC   #$78        $3A + $78 = $B2, d.h. Token für REM
D5E5-   BNE   $D56D       stimmt nicht, weiter kodieren
D5E7-   STA   $0E         REM erkannt, bis Endmarke (A=0!) unkodiert übernehmen
D5E9-   LDA   $0200,X     hole neues Zeichen
D5EC-   BEQ   $D5CD       Zeilenende, Routine abschließen
D5EE-   CMP   $0E         gemerktes Anführungszeichen? (vgl $D57A)
D5F0-   BEQ   $D5CD       ja, von nun ab wieder kodieren
D5F2-   INY               sonst weiter unkodiert übernehmen, bis 0 oder "
D5F3-   STA   $01FB,Y
D5F6-   INX
D5F7-   BNE   $D5E9       unbedingter Sprung

Tabellenzeiger auf nächstes Token setzen
D5F9-   LDX   $B8         hole Zeiger für Eingabe-Zeile
D5FB-   INC   $0F         Tokenzähler erhöhen
D5FD-   LDA   ($9D),Y     hole Zeichen aus Tabelle
D5FF-   INY               Zeiger erhöhen
D600-   BNE   $D604
D602-   INC   $9E
D604-   ASL               Bit7 -> C  (gesetzt, falls letztes Zeichen)
D605-   BCC   $D5FD       weiter
D607-   LDA   ($9D),Y     1. Zeichen des nächsten Befehls in der Tabelle
D609-   BNE   $D5A8       prüfen, ob es dieser Befehl ist
D60B-   LDA   $0200,X     kein Befehl passt, Zeichen unkodiert übernehmen
D60E-   BPL   $D5CB       und mit nächstem Zeichen weiter machen

Routine abschliessen
D610-   STA   $01FD,Y     Endmarke eintragen
D613-   DEC   $B9         CHRGET-Zeiger auf $01FF setzen
D615-   LDA   #$FF
D617-   STA   $B8
D619-   RTS
--------------------

Suche Programmzeile mit gegebener Zeilennummer
I: 50,51  Zeilennummer
O: gefunden -> C=1; 9B,9C zeigt auf Zeilenanfang
   sonst  C=0; 9B,9C zeigt auf nächste Zeile
D61A-   LDA   $67         Zeiger auf erste Programmzeile
D61C-   LDX   $68
D61E-   LDY   #$01
D620-   STA   $9B         nach 9B,9C
D622-   STX   $9C
D624-   LDA   ($9B),Y     Linkadresse, HByte
D626-   BEQ   $D647       Programm-Ende. Zurück mit C=0
```

← Data

← Rem

```
D628-   INY
D629-   INY
D62A-   LDA   $51          gesuchte Zeilennummer, HByte
D62C-   CMP   ($9B),Y      gefunden?
D62E-   BCC   $D648        gesuchte Zeile existiert nicht
D630-   BEQ   $D635        HByte stimmt, vergleiche LByte
D632-   DEY                stimmt nicht, weiter bei nächster Zeile
D633-   BNE   $D63E        unbedingter Sprung
D635-   LDA   $50          gesuchte Zeilennummer, LByte
D637-   DEY
D638-   CMP   ($9B),Y
D63A-   BCC   $D648        existiert nicht, zurück mit C=0
D63C-   BEQ   $D648        gesuchte Zeile gefunden, C=1
D63E-   DEY
D63F-   LDA   ($9B),Y      setze Zeiger auf Anfang der nächsten Zeile
D641-   TAX
D642-   DEY
D643-   LDA   ($9B),Y
D645-   BCS   $D61E        unbedingter Sprung
D647-   CLC                Einsprung, falls Zeile nicht gefunden
D648-   RTS
---------------------

Applesoft-Routine  NEW
D649-   BNE   $D648        nur durchführen, falls Endmarke folgt
D64B-   LDA   #$00
D64D-   STA   $D6          lösche Autostart-Flag
D64F-   TAY
D650-   STA   ($67),Y      setze 2 Byte als Programm-Endmarke  ( = 0) auf
D652-   INY                Programm-Anfang (normal $0801,$0802)
D653-   STA   ($67),Y
D655-   LDA   $67          setze Anfang der Variablen und Programm-Ende +1
D657-   ADC   #$02         jeweils auf Programm-Anfang +2
D659-   STA   $69
D65B-   STA   $AF
D65D-   LDA   $68
D65F-   ADC   #$00
D661-   STA   $6A
D663-   STA   $B0
D665-   JSR   $D697        setze CHRGET-Zeiger auf Programm-Anfang -1
D668-   LDA   #$00         CLEAR durchführen
---------------------

Applesoft-Routine  CLEAR
D66A-   BNE   $D696        nur durchführen, falls Endzeichen folgt
D66C-   LDA   $73          unteres String-Ende auf HIMEM setzen,
D66E-   LDY   $74          d.h. sämtliche Strings löschen
D670-   STA   $6F
D672-   STY   $70
D674-   LDA   $69          Feldvariablen-Anfangszeiger und Variablen-
D676-   LDY   $6A          Endezeiger := Variablen-Anfangszeiger,
D678-   STA   $6B          d.h. sämtliche Variablen löschen
D67A-   STY   $6C
D67C-   STA   $6D
D67E-   STY   $6E
D680-   JSR   $D849        RESTORE durchführen

Stack initialisieren
D683-   LDX   #$55
D685-   STX   $52          Deskriptoren-Stack initialisieren
D687-   PLA                rette Rücksprungadresse
```

```
D688-   TAY
D689-   PLA
D68A-   LDX   #$F8
D68C-   TXS                 Stack initialisieren
D68D-   PHA                 Rücksprungadresse wieder auf Stack
D68E-   TYA
D68F-   PHA
D690-   LDA   #$00
D692-   STA   $7A           CONT unmöglich machen
D694-   STA   $14           normale Variablenbehandlung
D696-   RTS
--------------------

CHRGET-Zeiger auf Programmstart -1 setzen
D697-   CLC
D698-   LDA   $67           67,68 zeigt auf Programmstart
D69A-   ADC   #$FF          subtrahiere 1
D69C-   STA   $B8
D69E-   LDA   $68
D6A0-   ADC   #$FF          korrigiere HByte, falls nötig
D6A2-   STA   $B9
D6A4-   RTS
--------------------

Applesoft-Routine  LIST
D6A5-   BCC   $D6B1         Ziffer folgt, d.h. Anfangswert angegeben
D6A7-   BEQ   $D6B1         gesamtes Programm auflisten
D6A9-   CMP   #$C9          "-" ?
D6AB-   BEQ   $D6B1         Endpunkt angegeben oder bis Programm-Ende
D6AD-   CMP   #$2C          "," ?
D6AF-   BNE   $D696 RTS     falsche Syntax, LIST nicht durchführen
D6B1-   JSR   $DA0C         hole Zeilennr. aus Programmtext, falls C=0
D6B4-   JSR   $D61A         suche zugehörige Zeile
D6B7-   JSR   $00B7         nächstes Befehlszeichen
D6BA-   BEQ   $D6CC         Endmarke
D6BC-   CMP   #$C9          "-" oder "," ?
D6BE-   BEQ   $D6C4
D6C0-   CMP   #$2C
D6C2-   BNE   $D648         nein, falsche Syntax, -> Return
D6C4-   JSR   $00B1         neues Befehlszeichen
D6C7-   JSR   $DA0C         falls Ziffer, Zeilennr. aus Programmtext holen
D6CA-   BNE   $D696         falls letztes Zeichen keine Endmarke, -> Return
D6CC-   PLA                 Rücksprungadresse vom Stack entfernen
D6CD-   PLA
D6CE-   LDA   $50           50,51 = Zeilennr., bis zu der gelistet wird
D6D0-   ORA   $51           ( beide null, falls nicht angegeben)
D6D2-   BNE   $D6DA
D6D4-   LDA   #$FF          da nicht angegeben, auf Maximalwert setzen
D6D6-   STA   $50
D6D8-   STA   $51
D6DA-   LDY   #$01
D6DC-   LDA   ($9B),Y       Linkadresse, HByte
D6DE-   BEQ   $D724         Programmende erreicht, Routine abschliessen
D6E0-   JSR   $D858         prüfe, ob Ctrl C gedrückt wurde
D6E3-   JSR   $DAFB         CR drucken
D6E6-   INY
D6E7-   LDA   ($9B),Y       aktuelle Zeilennummer nach X,A
D6E9-   TAX
D6EA-   INY
D6EB-   LDA   ($9B),Y
D6ED-   CMP   $51           mit Bereichsende vergleichen
```

```
D6EF-   BNE   $D6F5
D6F1-   CPX   $50
D6F3-   BEQ   $D6F7
D6F5-   BCS   $D724      überschritten, fertig
D6F7-   STY   $85        Index retten
D6F9-   JSR   $ED24      Zeilennummer drucken
D6FC-   LDA   #$20       Leerzeichen
D6FE-   LDY   $85        Index wieder holen
D700-   AND   #$7F
D702-   JSR   $DB5C      Ascii-Zeichen im Accu drucken
D705-   LDA   $24        Cursor-Spalte
D707-   CMP   #$21
D709-   BCC   $D712      kleiner als 33
D70B-   JSR   $DAFB      sonst neue Zeile (CR drucken)
D70E-   LDA   #$05
D710-   STA   $24        und Cursor in Spalte 5 setzen
D712-   INY
D713-   LDA   ($9B),Y    nächstes Zeichen
D715-   BNE   $D734      nicht Zeilenende, --> zur Druckroutine
D717-   TAY              =0
D718-   LDA   ($9B),Y    Linkadresse (= Anfang der nächsten Zeile)
D71A-   TAX
D71B-   INY
D71C-   LDA   ($9B),Y
D71E-   STX   $9B        auf Zeiger übertragen
D720-   STA   $9C
D722-   BNE   $D6DA      weiter mit dieser neuen Zeile (immer)
--------------------
D724-   LDA   #$0D       Ascii für CR
D726-   JSR   $DB5C      ausgeben
D729-   JMP   $D7D2      weiter in Interpreter-Hauptschleife
--------------------

Hole neues Zeichen aus Applesoft-Befehlstabelle
D72C-   INY              Zeiger um 1 erhöhen
D72D-   BNE   $D731
D72F-   INC   $9E
D731-   LDA   ($9D),Y    Zeichen laden
D733-   RTS
--------------------

Druckroutine für  LIST
D734-   BPL   $D702      kein Token, daher drucken und weiter
D736-   SEC              Token
D737-   SBC   #$7F       minus 127
D739-   TAX              als Zähler Suchschleife
D73A-   STY   $85        Zeiger für Programmzeile retten
D73C-   LDY   #$D0       Zeiger auf Anfang der Befehlstabelle
D73E-   STY   $9D
D740-   LDY   #$CF
D742-   STY   $9E
D744-   LDY   #$FF       (9D),Y zeigt nun auf $D0CF
D746-   DEX              nächster Befehl (0, falls richtiger Befehl)
D747-   BEQ   $D750      ist der Fall, Befehl ausdrucken
D749-   JSR   $D72C      hole nächstes Zeichen
D74C-   BPL   $D749      weiter, bis Befehlsende gefunden
D74E-   BMI   $D746      weiter mit nächstem Befehl
D750-   LDA   #$20       Leerzeichen
D752-   JSR   $DB5C      drucken
D755-   JSR   $D72C      neues Zeichen aus Befehlstabelle
D758-   BMI   $D75F      Bit7=1, Befehl fertig
```

```
D75A-   JSR   $DB5C      Zeichen drucken
D75D-   BNE   $D755      weiter mit nächstem Zeichen  (immer)
D75F-   JSR   $DB5C      letztes Zeichen drucken
D762-   LDA   #$20       Leerzeichen
D764-   BNE   $D6FE      weiter im Listing  (immer)
--------------------

Applesoft-Routine  FOR
D766-   LDA   #$80
D768-   STA   $14        verbiete Integervariable als Laufvariable
D76A-   JSR   $DA46      LET weist Laufvariable Anfangswert zu
D76D-   JSR   $D365      prüfe, ob schon eine Schleife mit dieser Laufvar.
D770-   BNE   $D777      nein
D772-   TXA              Stackpointer auf alte Schleife
D773-   ADC   #$0F       entferne alte Schleife vom Stack
D775-   TAX
D776-   TXS
D777-   PLA              Rücksprungadresse entfernen
D778-   PLA
D779-   LDA   #$09
D77B-   JSR   $D3D6      Platz für 18 Byte auf dem Stack? ja: hier weiter
D77E-   JSR   $D9A3      suche nächsten Befehl (Abstand kommt in Y)
D781-   CLC              Startadresse des Schleifenkörpers berechnen:
D782-   TYA              Abstand
D783-   ADC   $B8        zu aktuellem Zeiger addieren
D785-   PHA              und auf Stack schieben
D786-   LDA   $B9
D788-   ADC   #$00
D78A-   PHA
D78B-   LDA   $76        aktuelle Zeilennummer auf Stack schieben
D78D-   PHA
D78E-   LDA   $75
D790-   PHA
D791-   LDA   #$C1 B1    Token für TO
D793-   JSR   $DEC0      Syntax-Prüfung ob TO folgt
D796-   JSR   $DD6A      Prüfung, ob Laufvariable numerisch
D799-   JSR   $DD67      Ausdruck nach TO auswerten, muß numerisch sein
D79C-   LDA   $A2        Vorzeichen von FAC1  (Bit7=0 falls + )
D79E-   ORA   #$7F       in  MSB von FAC1 eintragen
D7A0-   AND   $9E
D7A2-   STA   $9E
D7A4-   LDA   #$AF       Adresse für indirekten Sprung:  $D7AF
D7A6-   LDY   #$D7
D7A8-   STA   $5E        merken
D7AA-   STY   $5F
D7AC-   JMP   $DE20      FAC1 (Schleifenendwert) auf Stack, dann hier weiter
D7AF-   LDA   #$13       Zeiger auf FP-Konstante = 1
D7B1-   LDY   #$E9
D7B3-   JSR   $EAF9      1 als Standart-STEP-Wert auf Stack
D7B6-   JSR   $00B7      neues Befehlszeichen
D7B9-   CMP   #$C7 B7    Token für STEP?
D7BB-   BNE   $D7C3      nein, STEP-Wert bleibt 1
D7BD-   JSR   $00B1      hole neues Zeichen
D7C0-   JSR   $DD67      STEP-Ausdruck auswerten, muß numerisch sein
D7C3-   JSR   $EB82      Vorzeichen nach Accu
D7C6-   JSR   $DE15      FAC1 (STEP-Wert) und VZ auf Stack
D7C9-   LDA   $86        Zeiger auf Laufvariable 85,86
D7CB-   PHA              auf Stack
D7CC-   LDA   $85
D7CE-   PHA
D7CF-   LDA   #$81       Kennzeichen für Schleifenparameter
D7D1-   PHA              auf Stack
--------------------
```

```
Interpreter-Hauptschleife (Befehlsausführung)
D7D2-   TSX                 Stackpointer merken
D7D3-   STX   $F8
D7D5-   JSR   $D858         Ctrl C gedrückt?  Falls nein, hier weiter
D7D8-   LDA   $B8           B8,B9 zeigt auf Zeichen hinter letztem Befehl
D7DA-   LDY   $B9
D7DC-   LDX   $76           = 255, falls Direktmodus
D7DE-   INX
D7DF-   BEQ   $D7E5         Direktmodus
D7E1-   STA   $79           Zeiger auf aktuellen Befehl -1 retten
D7E3-   STY   $7A
D7E5-   LDY   #$00
D7E7-   LDA   ($B8),Y       hole dieses Zeichen, muß Trennzeichen sein
D7E9-   BNE   $D842         nicht Zeilenende, Zeichen muß ":" sein
D7EB-   LDY   #$02          Zeilenende, prüfe, ob Programmende
D7ED-   LDA   ($B8),Y       Linkadresse HByte
D7EF-   CLC
D7F0-   BEQ   $D826         Programmende --> Warmstart  (C=0!)
D7F2-   INY                 sonst neue Zeilennummer holen
D7F3-   LDA   ($B8),Y
D7F5-   STA   $75           und in Zeilenzeiger eintragen
D7F7-   INY
D7F8-   LDA   ($B8),Y
D7FA-   STA   $76
D7FC-   TYA                 Abstand
D7FD-   ADC   $B8           zu CHRGET-Zeiger addieren
D7FF-   STA   $B8
D801-   BCC   $D805
D803-   INC   $B9

Ersteinsprung bei Direktmodus  (vgl $D459)
D805-   BIT   $F2           prüfe TRACE-Flag
D807-   BPL   $D81D         kein TRACE
D809-   LDX   $76           = 255, falls im Direktmodus
D80B-   INX
D80C-   BEQ   $D81D         kein TRACE, da Direktmodus
D80E-   LDA   #$23          Ascii für "#"
D810-   JSR   $DB5C         drucken
D813-   LDX   $75           enthält aktuelle Zeilennummer
D815-   LDA   $76
D817-   JSR   $ED24         Zeilennummer drucken
D81A-   JSR   $DB57         Leerzeichen drucken
D81D-   JSR   $00B1         hole Befehlszeichen
D820-   JSR   $D828         Applesoft-Routine aufrufen
D823-   JMP   $D7D2         weiter in Interpreter-Hauptschleife
--------------------
D826-   BEQ   $D88A         Sprung zu Warmstart  (vgl $D7F0)
--------------------

Applesoft-Routine aufrufen
I: Accu enthält Token
D828-   BEQ   $D857         Endzeichen --> fertig
D82A-   SBC   #$80
D82C-   BCC   $D83F         kein Token, kann nur LET-Anweisung sein
D82E-   CMP   #$40 2F       Token mit $C0 vergleichen
D830-   BCS   $D846         kein primärer Befehl --> SYNTAX ERROR
D832-   ASL                 sonst Token verdoppeln
D833-   TAY                 als Zeiger für Sprungtabelle
D834-   LDA   $D001,Y       Sprungtabelle enthält Adresse-1 der aufzurufende
D837-   PHA                 Routine  --> Stack
D838-   LDA   $D000,Y
```

```
D83B-   PHA
D83C-   JMP   $00B1       hole Zeichen nach Befehl, mit RTS zur Routine
---------------------
D83F-   JMP   $DA46       Sprung zur LET-Routine  (vgl. $D82C)
---------------------
D842-   CMP   #$3A        Zeichen = ":"?  (vgl. $D7E9)
D844-   BEQ   $D805       ja, neue Routine aufrufen
D846-   JMP   $DEC9       sonst SYNTAX ERROR
---------------------

Applesoft-Routine  RESTORE
D849-   SEC
D84A-   LDA   $67         67,68 = Anfang Programmtext
D84C-   SBC   #$01        minus 1
D84E-   LDY   $68
D850-   BCS   $D853
D852-   DEY               HByte korrigieren
D853-   STA   $7D         als DATA-Zeiger eintragen
D855-   STY   $7E
D857-   RTS
---------------------

Prüft, ob Ctrl C gedrückt wurde
D858-   LDA   $C000       Tastatur-Register
D85B-   CMP   #$83        Ctrl C gedrückt?
D85D-   BEQ   $D860       ja, STOP durchführen
D85F-   RTS               sonst weiter
---------------------

Extra Einsprung  STOP/END-Routine
D860-   JSR   $D553       Zeichen von Tastatur holen (Bit7=0)
D863-   LDX   #$FF        als Fehlerkode
D865-   BIT   $D8         ON ERR -Flag gesetzt?
D867-   BPL   $D86C       nein
D869-   JMP   $F2E9       ONERR-Fehlerbehandlung
D86C-   CMP   #$03        C=1, falls Ctrl C oder größer

Applesoft-Routine  STOP
I: C=1 (falls keine Ziffer folgt)
D86E-   BCS   $D871

Applesoft-Routine  END
D870-   CLC
D871-   BNE   $D8AF       es folgt kein Endezeichen, Routine nicht durchführen
D873-   LDA   $B8         Zeiger auf Trennzeichen vor letztem Befehl
D875-   LDY   $B9
D877-   LDX   $76         =255 falls Direktmodus
D879-   INX
D87A-   BEQ   $D888       Direktmodus
D87C-   STA   $79         letzten Befehl
D87E-   STY   $7A
D880-   LDA   $75         und letzte Zeilennummer retten
D882-   LDY   $76
D884-   STA   $77
D886-   STY   $78
D888-   PLA               Rücksprungadresse entfernen
D889-   PLA
D88A-   LDA   #$5D        Zeiger auf Text "BREAK"
D88C-   LDY   #$D3
D88E-   BCC   $D893       C=0 falls END
D890-   JMP   $D431       STOP: BREAK-Meldung drucken, dann Warmstart
```

```
D893-   JMP   $D43C      END:  Warmstart
---------------------

Applesoft-Routine  CONT
D896-   BNE   $D8AF      Nächstes Zeichen kein Trennzeichen, --> RTS
D898-   LDX   #$D2       Kode für CAN'T CONTINUE ERROR
D89A-   LDY   $7A        =0 nach CLEAR (vgl. $D692), sonst geretteter Zeiger
D89C-   BNE   $D8A1      CONT durchführen
D89E-   JMP   $D412      CAN'T CONTINUE ERROR
D8A1-   LDA   $79        geretteter Zeiger vor letzten Befehl, LByte
D8A3-   STA   $B8        -->  CHRGET-Zeiger
D8A5-   STY   $B9
D8A7-   LDA   $77        Nummer der unterbrochenen Zeile
D8A9-   LDY   $78
D8AB-   STA   $75        aktuelle Zeilennummer
D8AD-   STY   $76
D8AF-   RTS
---------------------

Applesoft-Routine  SAVE
D8B0-   SEC
D8B1-   LDA   $AF        Programmtext-Ende
D8B3-   SBC   $67        minus Programmtext-Anfang
D8B5-   STA   $50        ergibt Länge des Programmtextes
D8B7-   LDA   $B0
D8B9-   SBC   $68
D8BB-   STA   $51        Länge in 50,51
D8BD-   JSR   $D8F0      setze Monitor-Register
D8C0-   JSR   $FECD      Monitorroutine SAVE speichert Längendaten
D8C3-   JSR   $D901      setze Monitor-Register für Programmtext
D8C6-   JMP   $FECD      Monitorroutine SAVE speichert Programmtext
---------------------

Applesoft-Routine  LOAD
D8C9-   JSR   $D8F0      setze Monitor-Register für Längendaten
D8CC-   JSR   $FEFD      Mon. LOAD bringt Länge nach 50,51
D8CF-   CLC
D8D0-   LDA   $67        Programmtext-Anfang
D8D2-   ADC   $50        plus Länge
D8D4-   STA   $69        ergibt Programmtext-Ende +1
D8D6-   LDA   $68
D8D8-   ADC   $51
D8DA-   STA   $6A
D8DC-   LDA   $52
D8DE-   STA   $D6        Autostart-Flag eintragen
D8E0-   JSR   $D901      setze Monitor-Register für Programmtext
D8E3-   JSR   $FEFD      Mon. LOAD lädt Programmtext
D8E6-   BIT   $D6        prüfe Autostart-Flag
D8E8-   BPL   $D8ED      nicht gesetzt
D8EA-   JMP   $D665      RUN durchführen
D8ED-   JMP   $D4F2      CLEAR durchführen, dann Warmstart
---------------------

Setze Monitor-Register für Längendaten
D8F0-   LDA   #$50       Zeiger auf $0050
D8F2-   LDY   #$00
D8F4-   STA   $3C        nach Monitor-Register A1L,A1H
D8F6-   STY   $3D
D8F8-   LDA   #$52       A,Y zeigt auf $0052
D8FA-   STA   $3E        nach Monitor-Register A2L,A2H
D8FC-   STY   $3F
```

```
D8FE-   STY   $D6        Y=0, d.h. lösche Autustart-Modus
D900-   RTS
---------------------

Setze Monitor-Register für Programmtext
D901-   LDA   $67        Programmtext-Anfang
D903-   LDY   $68
D905-   STA   $3C        --> A1L,A1H
D907-   STY   $3D
D909-   LDA   $69        Programmtext-Ende +1
D90B-   LDY   $6A
D90D-   STA   $3E        --> A2L,A2H
D90F-   STY   $3F
D911-   RTS
---------------------

Applesoft-Routine  RUN
D912-   PHP              rette Z-Flag  ( gesetzt, falls Trennzeichen folgt )
D913-   DEC   $76        lösche Direkt-Flag  (d.h. nun sicher <> 255)
D915-   PLP
D916-   BNE   $D91B      es folgt kein Trennzeichen --> bedingtes RUN
D918-   JMP   $D665      CHRGET setzen, CLEAR durchf., mit RTS nach $D7D2
D91B-   JSR   $D66C      CLEAR durchführen
D91E-   JMP   $D935      weiter mit GOTO
---------------------

Applesoft-Routine  GOSUB
D921-   LDA   #$03
D923-   JSR   $D3D6      6 Byte Platz auf Stack?  (5 werden benötigt)
D926-   LDA   $B9        CHRGET-Zeiger auf Stack
D928-   PHA
D929-   LDA   $B8
D92B-   PHA
D92C-   LDA   $76        aktuelle Zeilennummer auf Stack
D92E-   PHA
D92F-   LDA   $75
D931-   PHA
D932-   LDA   #$B0       Kennzeichen im Stack für RETURN-Parameter
D934-   PHA              auf Stack
D935-   JSR   $00B7      hole erste Ziffer des Sprungziels
D938-   JSR   $D93E      GOTO durchführen
D93B-   JMP   $D7D2      weiter in Interpreter-Hauptschleife
---------------------

Applesoft-Routine  GOTO
D93E-   JSR   $DA0C      Zeilennummer aus Programmtext nach 50,51
D941-   JSR   $D9A6      suche Zeilenende  (Abstand --> Y)
D944-   LDA   $76        aktuelle Zeilennummer, HByte
D946-   CMP   $51        anzuspringende Zeilennummer, HByte
D948-   BCS   $D955      Zielzeile nicht hinter aktueller Zeile
D94A-   TYA              Länge dieser Zeile
D94B-   SEC
D94C-   ADC   $B8        addieren, CHRGET zeigt nun auf nächste Zeile
D94E-   LDX   $B9
D950-   BCC   $D959
D952-   INX
D953-   BCS   $D959      suche ab aktueller Zeile nach Zielzeile
D955-   LDA   $67        suche ab Programm-Anfang nach Zielzeile
D957-   LDX   $68
D959-   JSR   $D61E      suche Zeile mit Zeilennummer in 50,51
D95C-   BCC   $D97C      nicht gefunden --> UNDEF'D STATEMENT ERROR
```

```
D95E-   LDA   $9B        sonst Startadresse
D960-   SBC   #$01       minus 1
D962-   STA   $B8        --> CHRGET-Zeiger
D964-   LDA   $9C
D966-   SBC   #$00
D968-   STA   $B9
D96A-   RTS              zurück zur Interpreter-Hauptschleife
--------------------

Applesoft-Routinen  RETURN, POP
D96B-   BNE   $D96A      nur durchführen, falls Trennzeichen folgt
D96D-   LDA   #$FF       verhindern, daß Laufvariable erkannt wird, falls
D96F-   STA   $85        offene FOR/NEXT-Schleife auf Stack
D971-   JSR   $D365      Stack untersuchen
D974-   TXS              entferne 2 Returnadressen und evtl. Schleifenparamet
D975-   CMP   #$B0       RETURN-Parameter?
D977-   BEQ   $D984      ja, weiter mit RETURN/POP
--------------------
D979-   LDX   #$16       Kode für RETURN WITHOUT GOSUB ERROR
D97B-   BIT   $5AA2      = LDX #$5A, d.h. Kode für UNDEF'D STATEMENT ERROR
D97E-   JMP   $D412      zur Fehlerverwaltung
--------------------
D981-   JMP   $DEC9      SYNTAX ERROR
--------------------
D984-   PLA              = B0
D985-   PLA              Zeilennummer LByte
D986-   CPY   #$42       Y= (Token - $80) x 2; d.h. POP?
D988-   BEQ   $D9C5      ja, POP durchführen
D98A-   STA   $75        alte Zeilennummer wieder herstellen
D98C-   PLA
D98D-   STA   $76
D98F-   PLA              alten CHRGET-Zeiger wieder herstellen
D990-   STA   $B8
D992-   PLA
D993-   STA   $B9
D995-   JSR   $D9A3      suche nächstes Trennzeichen  (Abstand --> Y)
D998-   TYA              Abstand
D999-   CLC
D99A-   ADC   $B8        addieren, d.h. CHRGET zeigt auf nächsten Befehl
D99C-   STA   $B8
D99E-   BCC   $D9A2
D9A0-   INC   $B9
D9A2-   RTS
--------------------

Suche nächstes Trennzeichen / Zeilenende
O: Abstand in Y

Einsprung für Trennzeichen ":" suchen
D9A3-   LDX   #$3A       Ascii für ":"
D9A5-   BIT   $00A2      --> LDX #$00;  Einsprung für Zeilenende suchen
D9A8-   STX   $0D        Suchmarke merken
D9AA-   LDY   #$00
D9AC-   STY   $0E        Zeilenende als alternative Suchmarke
D9AE-   LDA   $0E        Suchmarken vertauschen
D9B0-   LDX   $0D
D9B2-   STA   $0D        enthält alternatives Suchzeichen
D9B4-   STX   $0E        enthält primäres Suchzeichen
D9B6-   LDA   ($B8),Y    hole Zeichen aus Programmzeile
D9B8-   BEQ   $D9A2      Zeilenende, fertig
D9BA-   CMP   $0E        primäres Suchzeichen?
```

```
D9BC-   BEQ   $D9A2      ja, fertig
D9BE-   INY
D9BF-   CMP   #$22       Anführungszeichen?
D9C1-   BNE   $D9B6      nein, weitersuchen
D9C3-   BEQ   $D9AE      innerhalb String  ":" ignorieren
--------------------

POP durchführen
D9C5-   PLA              Zeilennummer HByte und
D9C6-   PLA              CHRGET-Zeiger vom Stack entfernen
D9C7-   PLA
D9C8-   RTS
--------------------

Applesoft-Routine  IF
D9C9-   JSR   $DD7B      Bedingung auswerten
D9CC-   JSR   $00B7      nächstes Zeichen nach Bedingung
D9CF-   CMP   #$AB 8E    Token für GOTO?
D9D1-   BEQ   $D9D8      ja
D9D3-   LDA   #$C4 B4    Token für THEN
D9D5-   JSR   $DEC0      muß folgen, sonst SYNTAX ERROR
D9D8-   LDA   $9D        Exponent von FAC1, nicht null falls Bed. erfüllt
D9DA-   BNE   $D9E1      Bedingung erfüllt

Einsprung Applesoft-Routine  REM
D9DC-   JSR   $D9A6      Zeilenende suchen
D9DF-   BEQ   $D998      CHRGET auf neue Zeile, dort weiter (immer springen)
D9E1-   JSR   $00B7      hole nächstes Zeichen
D9E4-   BCS   $D9E9      keine Ziffer
D9E6-   JMP   $D93E      Ziffer, weiter bei GOTO
D9E9-   JMP   $D828      zur Befehlsausführung
--------------------

Applesoft-Routine  ON
D9EC-   JSR   $E6F8      hole 1-Byte-Integer aus Programmtext --> A1
D9EF-   PHA              nächstes Zeichen (im Accu) auf Stack retten
D9F0-   CMP   #$B0 8F    Token für GOSUB?
D9F2-   BEQ   $D9F8      ja
D9F4-   CMP   #$AB 8E    Token für GOTO?
D9F6-   BNE   $D981      nein, --> SYNTAX ERROR
D9F8-   DEC   $A1        Index als Zähler
D9FA-   BNE   $DA00      gewünschte Sprungadresse noch nicht erreicht
D9FC-   PLA              Token
D9FD-   JMP   $D82A      GOTO / GOSUB ausführen
DA00-   JSR   $00B1      hole neues Zeichen
DA03-   JSR   $DA0C      um eine Sprungadresse weiterrücken
DA06-   CMP   #$2C       Sprungadresse gefolgt von "," ?
DA08-   BEQ   $D9F8      ja, weiter
DA0A-   PLA              sonst ON-Statement ignorieren
DA0B-   RTS
--------------------

Zeilennummer aus Programmtext holen
I: Accu = 1.Zeichen, C=0 falls Ziffer
O: 50,51 enthält Zeilennr., Accu nächstes Zeichen
DA0C-   LDX   #$00
DA0E-   STX   $50        50,51 löschen
DA10-   STX   $51
DA12-   BCS   $DA0B RTS  keine Ziffer, fertig
DA14-   SBC   #$2F       #$30 subtrahieren  (C=0!)  ergibt reine Ziffer
DA16-   STA   $0D        merken
```

```
DA18-   LDA   $51        HByte des Zwischenergebnisses
DA1A-   STA   $5E        in Hilfsregister für Multiplikation mit zehn
DA1C-   CMP   #$19       vergleichen mit 25
DA1E-   BCS   $D9F4      25 x 256 x 10 = 64000 --> zu groß, SYNTAX ERROR
DA20-   LDA   $50        Accu = Hilfsregister LByte, C=0
DA22-   ASL              x 2
DA23-   ROL   $5E
DA25-   ASL              x 2
DA26-   ROL   $5E
DA28-   ADC   $50        50,51 addieren
DA2A-   STA   $50
DA2C-   LDA   $5E
DA2E-   ADC   $51
DA30-   STA   $51        damit verfünffacht
DA32-   ASL   $50        verdoppeln
DA34-   ROL   $51        also insgesamt x10, d.h. um 1 Dezimale verschoben
DA36-   LDA   $50
DA38-   ADC   $0D        gemerkte Ziffer addieren
DA3A-   STA   $50
DA3C-   BCC   $DA40
DA3E-   INC   $51        HByte korrigieren falls nötig
DA40-   JSR   $00B1      hole nächstes Zeichen
DA43-   JMP   $DA12      weiter, falls Ziffer und kein Überlauf
--------------------

Applesoft-Routine  LET
DA46-   JSR   $DFE3      angegebene Variable suchen oder neu einrichten
DA49-   STA   $85        Zeiger auf diese Variable retten
DA4B-   STY   $86
DA4D-   LDA   #$D0       Token für "="
DA4F-   JSR   $DEC0      muß folgen, sonst SYNTAX ERROR
DA52-   LDA   $12        Flags für Integervariable und
DA54-   PHA
DA55-   LDA   $11        Stringvariable
DA57-   PHA              auf Stack
DA58-   JSR   $DD7B      Ausdruck auswerten
DA5B-   PLA              Bit7 = 1 falls Stringvariable angegeben wurde
DA5C-   ROL              Bit7 --> C
DA5D-   JSR   $DD6D      stimmen Ausdruck und Variable überein? nein -> ERROR
DA60-   BNE   $DA7A      Stringausdruck in Stringvariable eintragen
DA62-   PLA              Bit7 = 1 falls Integervariable angegeben wurde
DA63-   BPL   $DA77      FP-Variable
DA65-   JSR   $EB72      FAC1 runden
DA68-   JSR   $E10C      in INT umwandeln (in A0,A1)
DA6B-   LDY   #$00
DA6D-   LDA   $A0        Integer in Variable eintragen
DA6F-   STA   ($85),Y    85,86 dient als Zeiger auf die angegebene Variable
DA71-   INY
DA72-   LDA   $A1
DA74-   STA   ($85),Y
DA76-   RTS
--------------------
DA77-   JMP   $EB27      FAC1 in Variable eintragen ( 85,86 als Zeiger )
--------------------

Stringdeskriptor in Variable eintragen
I: A0,A1 zeigt auf Deskriptor; 85,86 auf Variable
O: 8C,8D zeigt auf Originaldeskriptor; Y=2
DA7A-   PLA              Einsprung von LET, Integerflag entfernen
DA7B-   LDY   #$02
DA7D-   LDA   ($A0),Y    Stringanfang im Stringbereich, d.h. >= 6F,70 ?
```

```
DA7F-   CMP   $70        vergleiche HByte
DA81-   BCC   $DA9A      nein, weiter
DA83-   BNE   $DA8C      ja, prüfe 2. Vorraussetzung
DA85-   DEY
DA86-   LDA   ($A0),Y
DA88-   CMP   $6F        vergleiche LByte
DA8A-   BCC   $DA9A      nicht im Stringbereich, weiter
DA8C-   LDY   $A1        Deskriptor im Variablenbereich, d.h. >= 69,6A ?
DA8E-   CPY   $6A        vergleiche HByte
DA90-   BCC   $DA9A      nein, weiter
DA92-   BNE   $DAA1      ja, String in Stringbereich übertragen
DA94-   LDA   $A0
DA96-   CMP   $69        vergleiche LByte
DA98-   BCS   $DAA1      String in Stringbereich übertragen
DA9A-   LDA   $A0        Zeiger auf Deskriptor
DA9C-   LDY   $A1
DA9E-   JMP   $DAB7      Deskriptor in angegebene Variable eintragen

String in Stringbereich übertragen
DAA1-   LDY   #$00
DAA3-   LDA   ($A0),Y    Stringlänge
DAA5-   JSR   $E3D5      Anfang Strings herabsetzen, neuer Deskr.-> 9D,9E,9F
DAA8-   LDA   $8C        Zeiger auf Originaldeskriptor
DAAA-   LDY   $8D
DAAC-   STA   $AB        auf Hilfszeiger übertragen
DAAE-   STY   $AC
DAB0-   JSR   $E5D4      String in Stringbereich kopieren
DAB3-   LDA   #$9D       Zeiger auf neuen Deskriptor
DAB5-   LDY   #$00

Deskriptor in Variable eintragen
DAB7-   STA   $8C        Zeiger auf gültigen Deskriptor
DAB9-   STY   $8D
DABB-   JSR   $E635      Deskriptor aus Deskr.-Stack entfernen, falls obenauf
DABE-   LDY   #$00       Deskriptor kopieren:
DAC0-   LDA   ($8C),Y    Länge
DAC2-   STA   ($85),Y
DAC4-   INY
DAC5-   LDA   ($8C),Y    Adr.L
DAC7-   STA   ($85),Y
DAC9-   INY
DACA-   LDA   ($8C),Y    Adr.H
DACC-   STA   ($85),Y
DACE-   RTS
---------------------
DACF-   JSR   $DB3D      String drucken
DAD2-   JSR   $00B7      hole neues Zeichen

Applesoft-Routine  PRINT
DAD5-   BEQ   $DAFB      Trennzeichen, CR ausgeben, fertig
DAD7-   BEQ   $DB02      extra Einsprung (vgl. $DB32): fertig falls Trennz.
DAD9-   CMP   #$C0       Token für TAB( ?
DADB-   BEQ   $DB16 NOP  ja, Cursor positionieren  (C=1)
DADD-   CMP   #$C3       Token für SPC( ?
DADF-   CLC
DAE0-   BEQ   $DB16 NOP  ja, Leerzeichen drucken  (C=0)
DAE2-   CMP   #$2C       Ascii für Komma?
DAE4-   CLC
DAE5-   BEQ   $DB03      ja, Tabulator setzen
DAE7-   CMP   #$3B       Ascii für ";" ?
DAE9-   BEQ   $DB2F      ja, falls Trennzeichen folgt kein CR ausgeben
```

```
DAEB-   JSR   $DD7B      nächsten Ausdruck auswerten
DAEE-   BIT   $11        Stringflag prüfen
DAF0-   BMI   $DACF      String drucken
DAF2-   JSR   $ED34      sonst FAC1 in String umwandeln
DAF5-   JSR   $E3E7      Stringparameter feststellen (Deskr.-Zeiger A0,A1)
DAF8-   JMP   $DACF      String drucken

CR ausgeben
DAFB-   LDA   #$0D       Ascii für CR
DAFD-   JSR   $DB5C      ausgeben
DB00-   EOR   #$FF       invertieren
DB02-   RTS

Cursor auf nächsten Tabulator setzen
DB03-   LDA   $24        Cursor-Spalte
DB05-   CMP   #$18       mit 24 vergleichen
DB07-   BCC   $DB0E      kleiner, bleibt in selber Zeile
DB09-   JSR   $DAFB      neue Zeile (CR)
DB0C-   BNE   $DB2F      weiter mit neuem Zeichen
DB0E-   ADC   #$10       nächstes Vielfaches von 16 bestimmen
DB10-   AND   #$F0
DB12-   STA   $24        Cursorposition setzen
DB14-   BCC   $DB2F      weiter mit neuem Zeichen (immer)

TAB / SPC  ausführen
DB16-   PHP              C retten  ( =0 bei SPC, =1 bei TAB )
DB17-   JSR   $E6F5      hole 8-Bit-Integer aus Progr.-Text  --> X
DB1A-   CMP   #$29       folgendes Zeichen ")" ?
DB1C-   BEQ   $DB21      ok, weiter
DB1E-   JMP   $DEC9      SYNTAX ERROR
DB21-   PLP              hole C
DB22-   BCC   $DB2B      SPC, Leerzeichen drucken
DB24-   DEX
DB25-   TXA              TAB-Wert
DB26-   SBC   $24        Cursor-Spalte abziehen
DB28-   BCC   $DB2F      Ziel links von aktueller Pos. --> TAB ignorieren
DB2A-   TAX              Differenz = Anzahl zu druckender Leerzeichen
DB2B-   INX
DB2C-   DEX              Schleife für Leerzeichen drucken
DB2D-   BNE   $DB35      weitere Leerzeichen
DB2F-   JSR   $00B1      fertig, weiter mit neuem Zeichen
DB32-   JMP   $DAD7      zurück in PRINT-Routine
DB35-   JSR   $DB57      Leerzeichen drucken
DB38-   BNE   $DB2C      (immer)
--------------------

String drucken
I: A,Y Stringanfang;  Endmarke 00 oder 22 (")
O: CHRGET zeigt auf Zeichen hinter String
DB3A-   JSR   $E3E7      String auswerten
DB3D-   JSR   $E600      Deskriptorenstack bereinigen (Stringanfang in 5E,5F)
DB40-   TAX              Länge des Strings
DB41-   LDY   #$00       Zeiger initialisieren
DB43-   INX
DB44-   DEX              Schleife zum Drucken
DB45-   BEQ   $DB02      fertig, --> RTS
DB47-   LDA   ($5E),Y    hole Zeichen
DB49-   JSR   $DB5C      drucken
DB4C-   INY
DB4D-   CMP   #$0D       CR ?
DB4F-   BNE   $DB44      nein, weiter
```

```
DB51-   JSR   $DB00      invertieren (?)
DB54-   JMP   $DB44      weiter mit nächstem Zeichen
--------------------

Drucken eines einzelnen Zeichens
I: DB57-> Leerzeichen, DB5A-> ?, DB5C-> Ascii im Accu
O: X,Y unverändert; A ebenfalls bis auf Bit7 = 0
DB57-   LDA   #$20       Ascii für Leerzeichen
DB59-   BIT   $3FA9      --> LDA #$3F, d.h. Ascii für "?"
DB5C-   ORA   #$80       setze Bit7=1
DB5E-   CMP   #$A0
DB60-   BCC   $DB64      Ctrl-Zeichen, Flash-Flag ignorieren
DB62-   ORA   $F3        FLASH-Flag (=#$40 bei FLASH, sonst 00)
DB64-   JSR   $FDED      drucken mit Monitorroutine COUT
DB67-   AND   #$7F       Bit7=0
DB69-   PHA              Zeichen retten
DB6A-   LDA   $F1        SPEED-Wert
DB6C-   JSR   $FCA8      entsprechend verzögern mit Monitorroutine WAIT
DB6F-   PLA              Zeichen wieder holen
DB70-   RTS
--------------------

Fehlerbehandlung bei INPUT / GET / READ
DB71-   LDA   $15        Eingabe-Flag: $0=INPUT, $40=GET, $98=READ
DB73-   BEQ   $DB87      Fehler bei Input
DB75-   BMI   $DB7B      Fehler bei READ: SYNTAX ERROR IN Datazeile
DB77-   LDY   #$FF       Fehler bei GET, simuliere Direktmodus
DB79-   BNE   $DB7F
DB7B-   LDA   $7B        DATA-Zeiger für Fehlermeldung
DB7D-   LDY   $7C
DB7F-   STA   $75        aktuelle Zeilennummer
DB81-   STY   $76        bei GET-Fehler = $FF
DB83-   JMP   $DEC9      SYNTAX ERROR
DB86-   PLA              extra Einsprung bei INPUT-Fehler (keine Eingabe)
DB87-   BIT   $D8        teste ONERR-Flag
DB89-   BPL   $DB90      nicht gesetzt
DB8B-   LDX   #$FE       Kode für INPUT-Fehler
DB8D-   JMP   $F2E9      zur Fehlerbehandlung
DB90-   LDA   #$EF       Zeiger auf Text  "?REENTER", CR
DB92-   LDY   #$DC
DB94-   JSR   $DB3A      Text drucken
DB97-   LDA   $79        gemerkten CHRGET-Zeiger
DB99-   LDY   $7A
DB9B-   STA   $B8        wiederherstellen
DB9D-   STY   $B9
DB9F-   RTS              INPUT-Anweisung erneut durchführen
--------------------

Applesoft-Routine  GET
DBA0-   JSR   $E306      falls Direktmodus  ILLEGAL DIRECT ERROR
DBA3-   LDX   #$01       Zeiger auf $0201
DBA5-   LDY   #$02
DBA7-   LDA   #$00
DBA9-   STA   $0201      null als Endmarke
DBAC-   LDA   #$40       Kode für GET im Eingabe-Flag
DBAE-   JSR   $DBEB      zur Haupteingaberoutine bei READ
DBB1-   RTS
--------------------

Applesoft-Routine  INPUT
DBB2-   CMP   #$22       folgt Anführungszeichen?
```

```
DBB4-   BNE   $DBC4      nein, kein Dialogstring
DBB6-   JSR   $DE81      String auswerten
DBB9-   LDA   #$3B       Ascii für ";"
DBBB-   JSR   $DEC0      muß folgen, sonst SYNTAX ERROR
DBBE-   JSR   $DB3D      String drucken
DBC1-   JMP   $DBC7      kein Fragezeichen ausgeben
DBC4-   JSR   $DB5A      "?" drucken
DBC7-   JSR   $E306      prüfe, ob Programm-Modus
DBCA-   LDA   #$2C       Ascii für Komma
DBCC-   STA   $01FF      vor Eingabepuffer setzen
DBCF-   JSR   $D52C      Eingabezeile Empfangen (Y,X zeigt auf $01FF)
DBD2-   LDA   $0200      hole 1. Zeichen
DBD5-   CMP   #$03       Ctrl C ?
DBD7-   BNE   $DBE9      nein, zur Haupteingaberoutine bei READ
DBD9-   JMP   $D863      Ctrl C verarbeiten, d.h. STOP durchführen
--------------------
DBDC-   JSR   $DB5A      "?" drucken
DBDF-   JMP   $D52C      Eingabezeile empfangen
--------------------

Applesoft-Routine READ
DBE2-   LDX   $7D        DATA-Zeiger
DBE4-   LDY   $7E
DBE6-   LDA   #$98       Kode für READ
DBE8-   BIT   $00A9      --> LDA #$00, d.h. Kode für INPUT

Haupt-Eingaberoutine
I: X,Y zeigt auf Eingabedaten, Accu = Eingabeflag
DBEB-   STA   $15        Eingabe-Flag eintragen
DBED-   STX   $7F        Eingabezeiger merken
DBEF-   STY   $80
DBF1-   JSR   $DFE3      Variable suchen, ggf. neu einrichten
DBF4-   STA   $85        Zeiger auf Variable
DBF6-   STY   $86
DBF8-   LDA   $B8        Programmzeiger
DBFA-   LDY   $B9
DBFC-   STA   $87        retten
DBFE-   STY   $88
DC00-   LDX   $7F        Eingabezeiger
DC02-   LDY   $80
DC04-   STX   $B8        --> CHRGET-Zeiger
DC06-   STY   $B9
DC08-   JSR   $00B7      hole Eingabezeichen (je nach Eingabezeiger)
DC0B-   BNE   $DC2B      kein Trennzeichen
DC0D-   BIT   $15        Eingabe-Flag
DC0F-   BVC   $DC1F      INPUT / READ  (zu wenig Daten vorhanden)
DC11-   JSR   $FD0C      GET, hole Zeichen über Monitorroutine RDKEY
DC14-   AND   #$7F       lösche Bit7
DC16-   STA   $0200      Zeichen eintragen
DC19-   LDX   #$FF       Zeiger auf $01FF setzen
DC1B-   LDY   #$01
DC1D-   BNE   $DC27      zur weiteren Auswertung
DC1F-   BMI   $DCA0      READ, weitere DATA suchen
DC21-   JSR   $DB5A      INPUT, "?" ausgeben
DC24-   JSR   $DBDC      Eingabezeile empfangen (mit 2. Fragezeichen)
DC27-   STX   $B8        CHRGET-Zeiger auf $01FF
DC29-   STY   $B9
DC2B-   JSR   $00B1      hole Datenzeichen
DC2E-   BIT   $11        prüfe String-Flag
DC30-   BPL   $DC63      numerische Variable

String als Daten
```

```
DC32-   BIT   $15      Eingabe-Flag
DC34-   BVC   $DC3F    READ oder INPUT
DC36-   INX
DC37-   STX   $B8      Zeiger in Eingabepuffer merken
DC39-   LDA   #$00
DC3B-   STA   $0D      0 als Endmarke für GET-Zeichen
DC3D-   BEQ   $DC4B    überspringen, da für INPUT / READ
DC3F-   STA   $0D      1. Zeichen vorläufig als Trennzeichen eintragen
DC41-   CMP   #$22     Anführungszeichen?
DC43-   BEQ   $DC4C    ja, --> $0D = #$22, $0E = #$22
DC45-   LDA   #$3A     sonst Ascii für ":"
DC47-   STA   $0D
DC49-   LDA   #$2C     und Ascii für "," als Trennzeichen
DC4B-   CLC            kein ", erstes Zeichen gehört zum String
DC4C-   STA   $0E
DC4E-   LDA   $B8      Datenzeiger
DC50-   LDY   $B9
DC52-   ADC   #$00     um 1 erhöhen, falls Anf.-Zeichen (dann C=1)
DC54-   BCC   $DC57
DC56-   INY
DC57-   JSR   $E3ED    String auswerten, Endmarken in 0D,0E
DC5A-   JSR   $E73D    CHRGET-Zeiger holen (Stringende)
DC5D-   JSR   $DA7B    Deskriptor in Variable eintragen
DC60-   JMP   $DC72    weiter in Programmzeile

numerische Daten
DC63-   PHA            rette 1. Datenzeichen
DC64-   LDA   $0200    setze Z-Flag
DC67-   BEQ   $DC99    Endzeichen, d.h. keine Eingabe erfolgt
DC69-   PLA            hole Zeichen wieder
DC6A-   JSR   $BC4A    String in FP-Konstante umwandeln (--> FAC1)
DC6D-   LDA   $12      Integer-Flag
DC6F-   JSR   $DA63    Wert in Variable eintragen  (vgl. LET)
DC72-   JSR   $00B7    hole Datenzeichen hinter letztem Datensatz
DC75-   BEQ   $DC7E    Trennzeichen
DC77-   CMP   #$2C     Komma?
DC79-   BEQ   $DC7E    ja, ok.
DC7B-   JMP   $DB71    sonst Fehlerbehandlung
DC7E-   LDA   $B8      CHRGET-Zeiger
DC80-   LDY   $B9
DC82-   STA   $7F      in Eingabe-Zeiger merken
DC84-   STY   $80
DC86-   LDA   $87      geretteten Programmzeiger
DC88-   LDY   $88
DC8A-   STA   $B8      wiederherstellen
DC8C-   STY   $B9
DC8E-   JSR   $00B7    hole Zeichen aus Programmzeile
DC91-   BEQ   $DCC6    Trennzeichen, Routine abschliessen
DC93-   JSR   $DEBE    prüfe, ob Komma folgt
DC96-   JMP   $DBF1    falls ja, weiter mit nächster Eingabe
-----------------------
DC99-   LDA   $15      Eingabe-Flag
DC9B-   BNE   $DC69    READ oder GET, weiter
DC9D-   JMP   $DB86    INPUT, Fehlerbehandlung
-----------------------

Suche DATA-Statement
DCA0-   JSR   $D9A3    suche nächstes Trennzeichen  (Offset --> Y)
DCA3-   INY
DCA4-   TAX            Z-Flag aktualisieren
DCA5-   BNE   $DCB9    nicht Zeilenende
```

```
DCA7-   LDX   #$2A        Kode für OUT OF DATA ERROR
DCA9-   INY
DCAA-   LDA   ($B8),Y     Linkadresse, HByte
DCAC-   BEQ   $DD0D       Programmende, --> OUT OF DATA ERROR
DCAE-   INY
DCAF-   LDA   ($B8),Y     Zeilennummer LByte
DCB1-   STA   $7B
DCB3-   INY
DCB4-   LDA   ($B8),Y     und HByte
DCB6-   INY
DCB7-   STA   $7C         in DATA-Pointer eintragen
DCB9-   LDA   ($B8),Y     hole erstes Token
DCBB-   TAX               in X merken
DCBC-   JSR   $D998       CHRGET-Zeiger um Y erhöhen
DCBF-   CPX   #$83        Token für DATA?
DCC1-   BNE   $DCA0       nein, nächstes DATA-Statement suchen
DCC3-   JMP   $DC2B       weiter bei unterbrochenem READ
--------------------

Eingaberoutine abschließen
DCC6-   LDA   $7F         Eingabezeiger holen
DCC8-   LDY   $80
DCCA-   LDX   $15         Eingabe-Flag
DCCC-   BPL   $DCD1       INPUT oder GET
DCCE-   JMP   $D853       nächstes DATA-Statement in DATA-Zeiger 7D,7E merken
DCD1-   LDY   #$00
DCD3-   LDA   ($7F),Y     hole nächstes Datenzeichen
DCD5-   BEQ   $DCDE       Endmarke, fertig.  (bei GET immer der Fall)
DCD7-   LDA   #$DF        sonst Zeiger auf Text  "?EXTRA IGNORED"
DCD9-   LDY   #$DC
DCDB-   JMP   $DB3A       Meldung ausgeben, fertig
--------------------
DCDE-   RTS
--------------------

DCDF-   3F 45 58 54 52 41 20          ?EXTRA
        49 47 4E 4F 52 45 44 0D 00    IGNORED

DCEF-   3F 52 45 45 4E 54 45 52 0D 00 ?REENTER
--------------------

Applesoft-Routine  NEXT
DCF9-   BNE   $DCFF       kein Trennzeichen, d.h. NEXT-Variable angegeben
DCFB-   LDY   #$00        in 86 kennzeichnen, daß keine Variable angegeben
DCFD-   BEQ   $DD02       überspringe Variablensuche
DCFF-   JSR   $DFE3       angegebene Variable suchen
DD02-   STA   $85         Zeiger auf Variable merken
DD04-   STY   $86
DD06-   JSR   $D365       suche im Stack nach NEXT-Parameter
DD09-   BEQ   $DD0F       gefunden
DD0B-   LDX   #$00        Kode für NEXT WITHOUT FOR ERROR
DD0D-   BEQ   $DD78       Fehlermeldung  verarbeiten
DD0F-   TXS               aktuelle Schleifenparameter oben auf Stack
DD10-   INX
DD11-   INX
DD12-   INX
DD13-   INX
DD14-   TXA               Zeiger auf STEP-Wert, LByte
DD15-   INX
DD16-   INX
DD17-   INX
DD18-   INX
```

```
DD19-   INX
DD1A-   INX
DD1B-   STX   $60        Zeiger auf TO-Wert, LByte
DD1D-   LDY   #$01       HByte für beide Zeiger  (Stack = Pagel)
DD1F-   JSR   $EAF9      STEP-Wert nach FAC1 übertragen
DD22-   TSX
DD23-   LDA   $0109,X    Vorzeichen des STEP-Wertes
DD26-   STA   $A2        nach Vorzeichen-Flag von FAC1
DD28-   LDA   $85        Zeiger auf Laufvariable
DD2A-   LDY   $86
DD2C-   JSR   $E7BE      Variablenwert + STEP-Wert = neuer Variablenwert
DD2F-   JSR   $EB27      Ergebnis (in FAC1) in Laufvariable eintragen
DD32-   LDY   #$01       Zeiger auf TO-Wert, HByte
DD34-   JSR   $EBB4      FAC1 mit TO-Wert vergleichen (Accu =1 bei >, FF bei < )
DD37-   TSX
DD38-   SEC
DD39-   SBC   $0109,X    minus Vorzeichen STEP-Wert ( =1 bei +, FF bei - )
DD3C-   BEQ   $DD55      TO-Wert überschritten, Schleife fertig
DD3E-   LDA   $010F,X    Zeilennummer des FOR-Befehls
DD41-   STA   $75        als aktuelle Zeilennummer übernehmen
DD43-   LDA   $0110,X
DD46-   STA   $76
DD48-   LDA   $0112,X    Programmzeiger hinter FOR-Befehl
DD4B-   STA   $B8        als CHRGET-Zeiger übernehmen
DD4D-   LDA   $0111,X
DD50-   STA   $B9
DD52-   JMP   $D7D2      zur Programmausführung

Schleife abschließen
DD55-   TXA              Schleifendaten aus Stack entfernen
DD56-   ADC   #$11
DD58-   TAX
DD59-   TXS
DD5A-   JSR   $00B7      hole nächstes Zeichen aus Programmzeile
DD5D-   CMP   #$2C       Komma?
DD5F-   BNE   $DD52      nein, NEXT-Befehl fertig, weiter im Programm
DD61-   JSR   $00B1      hole nächstes Zeichen
DD64-   JSR   $DCFF      weiter mit nächster NEXT-Variablen
```

Numerischen Ausdruck auswerten
I: CHRGET zeigt auf erstes Zeichen des Ausdrucks
O: Ergebnis in FAC1

```
DD67-   JSR   $DD7B      Ausdruck auswerten
```

prüfe, ob Ausdruck numerisch

```
DD6A-   CLC
DD6B-   BIT   $38        weiter bei DD6D
```

prüfe, ob Stringausdruck

```
DD6C-   SEC
DD6D-   BIT   $11        String-Flag
DD6F-   BMI   $DD74      String
DD71-   BCS   $DD76      kein String, soll String sein --> TYPE MISMATCH
DD73-   RTS              stimmt überein, ok.
DD74-   BCS   $DD73      String, soll auch so sein, ok.
DD76-   LDX   #$A3       Kode für TYPE MISMATCH ERROR
DD78-   JMP   $D412      Fehlermeldung
```

Auswertung eines beliebigen Ausdrucks

```
I: CHRGET zeigt auf erstes Zeichen
O: falls Ausdruck numerisch --> FAC1
   sonst zeigt A0,A1 auf Stringdeskriptor
   CHRGET zeigt hinter Ausdruck, String-Flag aktuell
DD7B-   LDX   $B8        CHRGET-Zeiger vor Ausdruck setzen
DD7D-   BNE   $DD81
DD7F-   DEC   $B9
DD81-   DEC   $B8
DD83-   LDX   #$00       als Marke: keine offene Operation mehr auf Stack
DD85-   BIT   $48        weiter bei  DD87
DD86-   PHA              Vergleichs-Flag auf Stack
DD87-   TXA              Prioritäts-Flag
DD88-   PHA              auf Stack
DD89-   LDA   #$01
DD8B-   JSR   $D3D6      genügend Platz auf Stack?
DD8E-   JSR   $DE60      Operand aus Programmtext holen
DD91-   LDA   #$00
DD93-   STA   $89        Vergleichs-Flag löschen
DD95-   JSR   $00B7      neues Zeichen holen
DD98-   SEC
DD99-   SBC   #$CF BF
DD9B-   BCC   $DDB4      Token < CF, keine Vergleichsoperation
DD9D-   CMP   #$03
DD9F-   BCS   $DDB4      Token >= D2, keine Vergleichsoperation
DDA1-   CMP   #$01       C=0 falls >-Vergleich
DDA3-   ROL
DDA4-   EOR   #$01       damit  > 1  /  =  2  /  <  4
DDA6-   EOR   $89        damit  >= 3  /  <> 5  /  <= 6  /  =,<<,>> 0
DDA8-   CMP   $89        mit altem Flag vergleichen
DDAA-   BCC   $DE0D      zwei gleiche Relationen --> SYNTAX ERROR
DDAC-   STA   $89        Flag merken
DDAE-   JSR   $00B1      neues Zeichen
DDB1-   JMP   $DD98      damit weiter
DDB4-   LDX   $89        Vergleichs-Flag
DDB6-   BNE   $DDE4      Vergl.-Operation, Parameter bestimmen
DDB8-   BCS   $DE35      Token für SGN oder folgende
DDBA-   ADC   #$07
DDBC-   BCC   $DE35      Token für STEP oder Befehl davor
DDBE-   ADC   $11        C=1, String-Flag = FF bei String
DDC0-   BNE   $DDC5      springen, da nicht Token für + oder kein String
DDC2-   JMP   $E597      sonst String-Verknüpfung durchführen
DDC5-   ADC   #$FF       Kode  korrigieren, da C=1 addiert wurde
DDC7-   STA   $5E        Token minus AA  bei  + - * / ^ AND OR
DDC9-   ASL              x 2
DDCA-   ADC   $5E        ergibt 3 * (Token - AA)
DDCC-   TAY              als Zeiger in Tabelle der arithmetischen Operationen
DDCD-   PLA              Priorität auf Stack
DDCE-   CMP   $D0B2,Y    mit Priorität der Operation vergleichen
DDD1-   BCS   $DE3A      Vorrang für Stack-Operation, --> Op. durchführen
DDD3-   JSR   $DD6A      String-Flag darf nicht gesetzt sein
DDD6-   PHA              Prioritäts-Flag zurück auf Stack
DDD7-   JSR   $DDFD      Op.-Routinenadr. und Operand auf Stack, weiter bei DD86

Einsprung bei Rückkehr von Operations-Routine
DDDA-   PLA              Priorität der nächsten Operation auf Stack
DDDB-   LDY   $87        Tabellenzeiger für soeben vollzogene Operation
DDDD-   BPL   $DDF6      war nicht die letzte Operation im Ausdruck
DDDF-   TAX              Z-Flag aktualisieren
DDE0-   BEQ   $DE38      keine offene Operation mehr, Auswertung beendet
DDE2-   BNE   $DE43      nächste offene Operation auf Stack durchführen

Vorbereitung für Vergleichs-Operationen
```

```
DDE4-   LSR   $11        String-Flag löschen, C=1 falls Flag gesetzt war
DDE6-   TXA              Vergleichs-Kode
DDE7-   ROL              verdoppeln, C --> Bit1 (gesetzt, falls Stringvergleich)
DDE8-   LDX   $B8        CHRGET-Zeiger um 1 verringern
DDEA-   BNE   $DDEE
DDEC-   DEC   $B9
DDEE-   DEC   $B8
DDF0-   LDY   #$1B       Tabellenzeiger für Vergleichs-Operation
DDF2-   STA   $89        modifizierten Vergleichs-Kode merken
DDF4-   BNE   $DDCD      unbedingter Sprung zum Prioritäts-Vergleich
---------------------
DDF6-   CMP   $D0B2,Y    vergl. Stack-Prior. mit Prior. der eben durchgef. Op.
DDF9-   BCS   $DE43      Vorrang für Stack-Operation, --> durchführen
DDFB-   BCC   $DDD6      Zwischenergebnis als Operand, offene Operation
---------------------

offene Operation auf Stack retten
DDFD-   LDA   $D0B4,Y    Sprungadresse für Op.-Routine aus Tabelle
DE00-   PHA              auf Stack retten
DE01-   LDA   $D0B3,Y
DE04-   PHA
DE05-   JSR   $DE10      Operand (in FAC1) auf Stack retten
DE08-   LDA   $89        Vergleichs-Flag laden
DE0A-   JMP   $DD86      nächstes Element des Ausdrucks auswerten
---------------------
DE0D-   JMP   $DEC9      SYNTAX ERROR durchführen
---------------------
DE10-   LDA   $A2        Vorzeichen von FAC1
DE12-   LDX   $D0B2,Y    Priorität der Operation nach X retten
DE15-   TAY              Vorzeichen nach Y
DE16-   PLA              Rücksprungadresse nach 5E,5F retten
DE17-   STA   $5E
DE19-   INC   $5E
DE1B-   PLA
DE1C-   STA   $5F
DE1E-   TYA              Vorzeichen von FAC1
DE1F-   PHA              auf Stack
DE20-   JSR   $EB72      FAC1 runden
DE23-   LDA   $A1        und auf Stack retten
DE25-   PHA
DE26-   LDA   $A0
DE28-   PHA
DE29-   LDA   $9F
DE2B-   PHA
DE2C-   LDA   $9E
DE2E-   PHA
DE2F-   LDA   $9D
DE31-   PHA
DE32-   JMP   ($005E)    Rücksprung zur aufrufenden Routine
---------------------

Auswertung abschließen
DE35-   LDY   #$FF       Kennzeichen, daß Ende des Ausdrucks erreicht
DE37-   PLA              Priorität von Stack
DE38-   BEQ   $DE5D      keine offene Operation mehr, Auswertung beendet
DE3A-   CMP   #$64       offene Vergleichsoperation?
DE3C-   BEQ   $DE41      ja, Operand darf String sein
DE3E-   JSR   $DD6A      Operand muß numerisch sein
DE41-   STY   $87        Tabellenzeiger merken

Operation auf Stack durchführen
```

```
DE43-   PLA                modifiziertes Vergleichs-Flag
DE44-   LSR                in Normalform bringen, C=1 falls String-Vergleich
DE45-   STA   $16          Vergleichsflag merken
DE47-   PLA                Operand von Stack nach FAC2 übertragen
DE48-   STA   $A5          Exponent
DE4A-   PLA
DE4B-   STA   $A6          Mantissel
DE4D-   PLA
DE4E-   STA   $A7          Mantisse2
DE50-   PLA
DE51-   STA   $A8          Mantisse3 (falls String als Operand zeigt A8,A9
DE53-   PLA                          auf Deskriptor)
DE54-   STA   $A9          Mantisse4
DE56-   PLA
DE57-   STA   $AA          Vorzeichen
DE59-   EOR   $A2          mit Vorzeichen von FAC1 verknüpfen
DE5B-   STA   $AB          ergibt Kombi-Vorzeichen
DE5D-   LDA   $9D          Exponent von FAC1 laden
DE5F-   RTS                Operation durchführen (bzw. Ausdruck fertig, zurück)
--------------------

Konstante oder Variable aus Programmtext holen
DE60-   LDA   #$00         String-Flag löschen
DE62-   STA   $11
DE64-   JSR   $00B1        hole erstes Zeichen
DE67-   BCS   $DE6C        keine Ziffer
DE69-   JMP   $EC4A        Zahlenstring in FP-Konstante umwandeln (FAC1), fertig
DE6C-   JSR   $E07D        prüfe ob Buchstabe, C=1 falls ja
DE6F-   BCS   $DED5        ja, d.h. Variable --> Variablenwert holen
DE71-   CMP   #$2E         Dezimalpunkt?
DE73-   BEQ   $DE69        ja, Zahlenwert nach FAC1
DE75-   CMP   #$C9  B9     Minuszeichen?
DE77-   BEQ   $DECE        ja, Vorzeichenwechsel als offene Operation auf Stack
DE79-   CMP   #$C8  B8     Pluszeichen?
DE7B-   BEQ   $DE64        ja, ignorieren
DE7D-   CMP   #$22         Anführungszeichen?
DE7F-   BNE   $DE90        nein, kein String
DE81-   LDA   $B8          String, CHRGET-Zeiger erhöhen
DE83-   LDY   $B9
DE85-   ADC   #$00
DE87-   BCC   $DE8A
DE89-   INY                CHRGET zeigt nun auf erstes Stringzeichen
DE8A-   JSR   $E3E7        String auswerten
DE8D-   JMP   $E73D        CHRGET-Zeiger (Stringende) holen, fertig
DE90-   CMP   #$C6  B6     Token für NOT?
DE92-   BNE   $DEA4        nein
DE94-   LDY   #$18         Tabellenzeiger für NOT
DE96-   BNE   $DED0        NOT als offene Operation auf Stack

NOT durchführen
DE98-   LDA   $9D          Exponent von FAC1 (=0, falls Bedingung nicht erfüllt)
DE9A-   BNE   $DE9F        Bedingung erfüllt, um zu negieren --> Y=0
DE9C-   LDY   #$01         Wahrheitswert war null, ergibt negiert Y=1
DE9E-   BIT   $00A0        --> DE9F-  LDY #$00
DEA1-   JMP   $E301        Integer in Y umwandeln in FP-Konstante (in FAC1)
--------------------
DEA4-   CMP   #$C2         Token für FN?
DEA6-   BNE   $DEAB  NOP   nein
DEA8-   JMP   $E354        sonst FN auswerten
DEAB-   CMP   #~~$D2~~  C2 Token für SGN?
DEAD-   BCC   $DEB2        keine Funktion, prüfe ob Klammerausdruck
```

```
DEAF-   JMP   $DF0C      zur Funktions-Auswertung
DEB2-   JSR   $DEBB      prüfe  ob "(", sonst SYNTAX ERROR  ← Löscha
DEB5-   JSR   $DD7B      Term auswerten

Hilfsroutine zur Syntax-Prüfung
DEB8-   LDA   #$29       Ascii für ")"
DEBA-   BIT   $28A9      --> DEBB-  LDA #$28 , d.h. Ascii für "("
DEBD-   BIT   $2CA9      --> DEBE-  LDA #$2C , d.h. Ascii für ","
DEC0-   LDY   #$00
DEC2-   CMP   ($B8),Y    mit zu prüfendem Zeichen vergleichen
DEC4-   BNE   $DEC9      stimmt nicht, SYNTAX ERROR
DEC6-   JMP   $00B1      ok, zurück mit neuem Zeichen
--------------------
DEC9-   LDX   #$10       Kode für SYNTAX ERROR
DECB-   JMP   $D412      zur Fehlermeldung
--------------------
DECE-   LDY   #$15       Tabellenzeiger für Vorzeichenwechsel
DED0-   PLA              entferne Rücksprungadresse
DED1-   PLA
DED2-   JMP   $DDD7      als offene Operation auf Stack
--------------------

Variablenwert holen
I: CHRGET zeigt auf Variablenname
O: Inhalt ,  falls numerisch, in FAC1
   sonst zeigt A0,A1 auf Stringdeskriptor
DED5-   JSR   $DFE3      Variable suchen, ggf. neu einrichten
DED8-   STA   $A0        Zeiger auf Variablen-Eintrag
DEDA-   STY   $A1
DEDC-   LDX   $11        String-Flag
DEDE-   BEQ   $DEE5      kein String
DEE0-   LDX   #$00
DEE2-   STX   $AC        ?
DEE4-   RTS
DEE5-   LDX   $12        Integer-Flag
DEE7-   BPL   $DEF6      FP-Variable, Wert nach FAC1
DEE9-   LDY   #$00
DEEB-   LDA   ($A0),Y    Integer nach A,Y
DEED-   TAX
DEEE-   INY
DEEF-   LDA   ($A0),Y
DEF1-   TAY
DEF2-   TXA
DEF3-   JMP   $E2F2      Integer in A,Y nach FAC1
--------------------
DEF6-   JMP   $EAF9      FP-Variable (A,Y als Zeiger) nach FAC1
--------------------

Applesoft-Routine  SCRN
DEF9-   JSR   $00B1      hole nächstes Zeichen
DEFC-   JSR   $F1EC      hole Plot-Parameter LORES-Grafik
DEFF-   TXA              2. Parameter
DF00-   LDY   $F0        1. Parameter
DF02-   JSR   $F871      Monitorroutine SCRN bringt Farbe des Punktes
DF05-   TAY              Farbkode
DF06-   JSR   $E301      nach FAC1 bringen
DF09-   JMP   $DEB8      ")" muß folgen, sonst SYNTAX ERROR
--------------------

Auswertung von Funktionen
DF0C-   CMP   #$D7       Token für SCRN?   ?
```

```
DF0E-   BEQ   $DEF9      ja, zur Routine
DF10-   ASL              Token x 2 mod 256
DF11-   PHA              retten
DF12-   TAX
DF13-   JSR   $00B1      neues Zeichen
DF16-   CPX   #$CF
DF18-   BCC   $DF3A      Token war < E8, d.h. nicht LEFT$/MID$/RIGHT$
DF1A-   JSR   $DEBB      sonst prüfen, ob "(" folgt
DF1D-   JSR   $DD7B      1. Ausdruck auswerten
DF20-   JSR   $DEBE      prüfen, ob Komma folgt
DF23-   JSR   $DD6C      prüfen, ob Stringausdruck
DF26-   PLA              Token x 2 mod 256
DF27-   TAX              in X merken
DF28-   LDA   $A1        Stringdeskriptor
DF2A-   PHA              auf Stack retten
DF2B-   LDA   $A0
DF2D-   PHA
DF2E-   TXA              "Token" retten
DF2F-   PHA
DF30-   JSR   $E6F8      1Byte-Integer (1.Parameter) aus Programmtext --> X
DF33-   PLA              "Token"
DF34-   TAY              als Tabellenzeiger nach Y
DF35-   TXA              1. Parameter
DF36-   PHA              auf Stack retten
DF37-   JMP   $DF3F      Routine anspringen
--------------------
DF3A-   JSR   $DEB2      Klammerausdruck auswerten
DF3D-   PLA              Token x 2 mod 256
DF3E-   TAY              als Tabellenzeiger
DF3F-   LDA   $CFDC,Y    Sprungadresse aus Tabelle
DF42-   STA   $91        nach 91,92
DF44-   LDA   $CFDD,Y
DF47-   STA   $92
DF49-   JSR   $0090      Routine aufrufen
DF4C-   JMP   $DD6A      Ergebnis muß numerisch sein, falls ok. fertig
--------------------

Logische Verknüpfung OR durchführen
DF4F-   LDA   $A5        1. Wahrheitswert
DF51-   ORA   $9D        oder 2. Wahrheitswert ungleich null?
DF53-   BNE   $DF60      ja, OR-Bed. erfüllt  (sonst AND sicher nicht erfüllt)

Logische Verknüpfung AND
DF55-   LDA   $A5        1. Wahrheitswert
DF57-   BEQ   $DF5D      null, d.h. nicht erfüllt
DF59-   LDA   $9D        2. Wahrheitswert
DF5B-   BNE   $DF60      erfüllt, damit AND-Bedingung erfüllt
DF5D-   LDY   #$00       neuer Wahrheitswert = null
DF5F-   BIT   $01A0      --> DF60-  LDY #$01, d.h. neuer Wahrheitswert = 1
DF62-   JMP   $E301      Wahrheitswert in Y nach FAC1
--------------------

Routine für Vergleichs-Operationen
I: C=1 falls erster Ausdruck String
DF65-   JSR   $DD6D      prüfe ob 2. Ausdruck gleicher Typ wie erster
DF68-   BCS   $DF7D      Vergleich zweier Strings

Vergleich zweier numerischer Größen
I: Größen in FAC1,FAC2
DF6A-   LDA   $AA        Vorzeichen FAC2
DF6C-   ORA   #$7F
```

```
DF6E-   AND   $A6        in höchste Mantissenstelle eintragen
DF70-   STA   $A6
DF72-   LDA   #$A5       Zeiger auf FAC2
DF74-   LDY   #$00
DF76-   JSR   $EBB2      FAC1 mit FAC2 vergleichen
DF79-   TAX              1,0,255 bei 1. Zahl  < = >  2. Zahl
DF7A-   JMP   $DFB0      Wahrheitswert der Vergleichsoperation feststellen
```

Vergleich zweier Strings
I: Zeiger auf Stringdeskriptoren in A0,A1 und A8,A9

```
DF7D-   LDA   #$00
DF7F-   STA   $11        String-Flag löschen
DF81-   DEC   $89
DF83-   JSR   $E600      unnützen String entfernen
DF86-   STA   $9D        Deskriptor für 2. String nach 9D,9E,9F
DF88-   STX   $9E
DF8A-   STY   $9F
DF8C-   LDA   $A8        Deskriptorzeiger für 1. String
DF8E-   LDY   $A9
DF90-   JSR   $E604      unnützen String entfernen
DF93-   STX   $A8        Zeiger auf 1. String
DF95-   STY   $A9
DF97-   TAX              Länge des 1. Strings
DF98-   SEC
DF99-   SBC   $9D        minus Länge des 2. Strings
DF9B-   BEQ   $DFA5      beide Strings gleich lang
DF9D-   LDA   #$01
DF9F-   BCC   $DFA5      1. String kürzer
DFA1-   LDX   $9D        X = Länge des kürzeren Strings
DFA3-   LDA   #$FF
DFA5-   STA   $A2        1,0,255 bei Länge 1  < = >  Länge 2
DFA7-   LDY   #$FF       als Index, ergibt Anfangswert 0
DFA9-   INX
DFAA-   INY              Vergleichs-Schleife
DFAB-   DEX
DFAC-   BNE   $DFB5      Ende des kürzeren Strings noch nicht erreicht
DFAE-   LDX   $A2        Längenvergleich
DFB0-   BMI   $DFC1      Länge 1 > Länge 2 ,  C=1
DFB2-   CLC
DFB3-   BCC   $DFC1      Länge 1 <= Länge2 ,  C=0
DFB5-   LDA   ($A8),Y    Zeichen aus 1. String
DFB7-   CMP   ($9E),Y    mit Zeichen aus 2. String vergleichen
DFB9-   BEQ   $DFAA      Übereinstimmung, vergleiche nächstes Zeichen
DFBB-   LDX   #$FF
DFBD-   BCS   $DFC1      1. Ausdruck >  2. Ausdruck
DFBF-   LDX   #$01
DFC1-   INX              X = 0,1,2  bei  1. Ausdruck  > = <  2. Ausdruck
DFC2-   TXA
DFC3-   ROL              A = 1,2,4
DFC4-   AND   $16        Bit 0,1,2 bedeutet  > = < in Vergleichs-Flag
DFC6-   BEQ   $DFCA      keine Übereinstimmung, —> Wahrheitswert = 0
DFC8-   LDA   #$01       als Wahrheitswert
DFCA-   JMP   $EB93      Wahrheitswert nach FAC1
```

Applesoft-Routine PDL

```
DFCD-   JSR   $E6FB      1Byte-Integer (Paddlenummer) aus Progr.Text holen
DFD0-   JSR   $FB1E      Paddle abfragen über Monitorroutine PDL
DFD3-   JMP   $E301      Ergebnis nach FAC1
```

Applesoft-Routine DIM

```
DFD6-   JSR   $DEBE      prüfe ob Komma folgt

Einsprung bei Aufruf
DFD9-   TAX              1. Namenbyte = Buchstabe, --> Bit6 = 1
DFDA-   JSR   $DFE8      Variable dimensionieren
DFDD-   JSR   $00B7      hole neues Zeichen
DFE0-   BNE   $DFD6      kein Trennzeichen, weitere Variable dimensionieren
DFE2-   RTS
--------------------

suche Variable
I: CHRGET zeigt auf 1. Namensbyte
O: A,Y = 83,84 zeigt auf Variablen-Eintragung
   9B,9C zeigt auf Variablenname
DFE3-   LDX   #$00       nicht von DIM gerufen
DFE5-   JSR   $00B7      hole 1. Zeichen
DFE8-   STX   $10        DIM-Flag, Bit6 = 1 falls DIM
DFEA-   STA   $81        Variablenname, 1. Byte
DFEC-   JSR   $00B7      hole selbes Zeichen nochmal
DFEF-   JSR   $E07D      prüfe ob Buchstabe
DFF2-   BCS   $DFF7      ja, ok
DFF4-   JMP   $DEC9      nein, SYNTAX ERROR
DFF7-   LDX   #$00
DFF9-   STX   $11        String-Flag löschen
DFFB-   STX   $12        Integer-Flag löschen
DFFD-   JMP   $E007      Entry-Vektoren überspringen
--------------------
E000-   JMP   $F128      Kaltstart
E003-   JMP   $D43C      Warmstart
E006-   BRK
--------------------
E007-   JSR   $00B1      hole neues Zeichen
E00A-   BCC   $E011      Ziffer
E00C-   JSR   $E07D      prüfe ob Buchstabe
E00F-   BCC   $E01C      nein
E011-   TAX              sonst merken (= Variablenname, 2. Byte)
E012-   JSR   $00B1      neues Zeichen holen
E015-   BCC   $E012      Ziffer, ignorieren da schon 2 gültige Zeichen
E017-   JSR   $E07D      prüfe ob Buchstabe
E01A-   BCS   $E012      ja, ignorieren da schon 2 gültige Zeichen
E01C-   CMP   #$24       Ascii für "$" ?
E01E-   BNE   $E026      nein
E020-   LDA   #$FF
E022-   STA   $11        String-Flag setzen
E024-   BNE   $E036      unbedingter Sprung
E026-   CMP   #$25       Ascii für "%" ?
E028-   BNE   $E03D      nein, also FP-Variable
E02A-   LDA   $14        Bit7 = 1 falls Integer-Variablen nicht erlaubt
E02C-   BMI   $DFF4      SYNTAX ERROR
E02E-   LDA   #$80
E030-   STA   $12        Integer-Flag setzen
E032-   ORA   $81        setze Bit7 in 1.Namensbyte, da Integervariable
E034-   STA   $81
E036-   TXA              2. Namensbyte
E037-   ORA   #$80       Bit7 setzen, da keine FP-Variable
E039-   TAX
E03A-   JSR   $00B1      neues Zeichen holen
E03D-   STX   $82        2. Namensbyte eintragen
E03F-   SEC
E040-   ORA   $14        = #$40, falls Feldvariable ohne "(", vgl STORE
E042-   SBC   #$28       minus Ascii für "(" ergibt 0, falls Feld und $14=0
```

```
E044-   BNE   $E049      nicht 0, prüfen ob von STORE aufgerufen
E046-   JMP   $E11E      --> Feldvariable suchen
E049-   BIT   $14        teste Variablen-Hilfsflag
E04B-   BMI   $E04F      >= $80, sicher nicht von STORE
E04D-   BVS   $E046      Bit6 = 1, d.h. von STORE gerufen --> Feld suchen
E04F-   LDA   #$00
E051-   STA   $14        Hilfs-Flag löschen

einfache Variable suchen
E053-   LDA   $69        zeigt auf Anfang der Variablen
E055-   LDX   $6A
E057-   LDY   #$00       Zeiger initialisieren
E059-   STX   $9C        Such-Zeiger setzen
E05B-   STA   $9B
E05D-   CPX   $6C        prüfe, ob A,X = 6B,6C  (d.h. Ende der einfachen
E05F-   BNE   $E065                               Variablen erreicht? )
E061-   CMP   $6B
E063-   BEQ   $E087      ja, Variable nicht gefunden --> neu einrichten
E065-   LDA   $81        1. Namensbyte
E067-   CMP   ($9B),Y    vergleichen
E069-   BNE   $E073      stimmt nicht überein, weiter mit nächster Variablen
E06B-   LDA   $82        sonst 2. Namensbyte
E06D-   INY
E06E-   CMP   ($9B),Y    vergleichen
E070-   BEQ   $E0DE      stimmt auch, Variable gefunden --> fertig
E072-   DEY
E073-   CLC              sonst Suchzeiger um 7 (= Länge der Variablen)
E074-   LDA   $9B        erhöhen
E076-   ADC   #$07
E078-   BCC   $E05B
E07A-   INX              HByte korrigieren falls nötig
E07B-   BNE   $E059      A,X zeigt nun auf nächste Variable, diese vergleichen
--------------------

prüfe, ob Zeichen im Accu ein Buchstabe ist
O: C=1 falls Buchstabe
E07D-   CMP   #$41       Ascii für "A"
E07F-   BCC   $E086      kleiner, kein Buchstabe
E081-   SBC   #$5B
E083-   SEC
E084-   SBC   #$A5       C=1, falls Accu kleiner als $5B war, d.h. Buchstabe
E086-   RTS
--------------------

einfache Variable neu einrichten
E087-   PLA              Rücksprungadresse, LByte nach Accu
E088-   PHA
E089-   CMP   #$D7       von $DDD7 aufgerufen, d.h. von Ausdruck-Auswertung?
E08B-   BNE   $E09C      nein, also Variable neu einrichten
E08D-   TSX
E08E-   LDA   $0102,X    Rücksprungadresse, HByte holen
E091-   CMP   #$DE       von $DED5 aufgerufen,d.h. Variablenwert-Bestimmung ?
E093-   BNE   $E09C      nein, also Variable neu einrichten
E095-   LDA   #$9A       nicht neu einrichten, da gesuchter Variable nichts
E097-   LDY   #$E0       zugewiesen wird. Zurück mit Zeiger auf FP-Konst = 0
E099-   RTS
--------------------
E09A-   00 00          FP-Konst. = 0 (nur Exp,MSB)
--------------------

Platz für neue einfache Variable schaffen
```

```
E09C-   LDA   $6B        Anfang der Feldvariablen
E09E-   LDY   $6C
E0A0-   STA   $9B        als alter Blockanfang
E0A2-   STY   $9C
E0A4-   LDA   $6D        Ende der Feldvariablen +1
E0A6-   LDY   $6E
E0A8-   STA   $96        als altes Blockende +1
E0AA-   STY   $97
E0AC-   CLC
E0AD-   ADC   #$07       plus 7
E0AF-   BCC   $E0B2
E0B1-   INY
E0B2-   STA   $94        als neues Blockende +1
E0B4-   STY   $95
E0B6-   JSR   $D393      Feldvariablen um 7 Byte verschieben
E0B9-   LDA   $94        neuer Blockanfang
E0BB-   LDY   $95
E0BD-   INY
E0BE-   STA   $6B        als neuen Anfang der Feldvariablen eintragen
E0C0-   STY   $6C

neue Variable initialisieren
E0C2-   LDY   #$00
E0C4-   LDA   $81        1. Namensbyte
E0C6-   STA   ($9B),Y    eintragen
E0C8-   INY
E0C9-   LDA   $82        2. Namensbyte
E0CB-   STA   ($9B),Y    eintragen
E0CD-   LDA   #$00
E0CF-   INY              Variablen-Eintragung nullsetzen
E0D0-   STA   ($9B),Y
E0D2-   INY
E0D3-   STA   ($9B),Y
E0D5-   INY
E0D6-   STA   ($9B),Y
E0D8-   INY
E0D9-   STA   ($9B),Y
E0DB-   INY
E0DC-   STA   ($9B),Y

Zeiger auf neue Variable setzen
E0DE-   LDA   $9B        Zeiger auf Variablennamen
E0E0-   CLC
E0E1-   ADC   #$02       um 2 erhöhen
E0E3-   LDY   $9C
E0E5-   BCC   $E0E8
E0E7-   INY
E0E8-   STA   $83        ergibt Zeiger auf Variablen-Eintragung
E0EA-   STY   $84
E0EC-   RTS
--------------------

Zeiger hinter Feldvariablen-Kopf setzen
E0ED-   LDA   $0F        Anzahl der Dimensionen
E0EF-   ASL              verdoppeln
E0F0-   ADC   #$05       und 5 addieren ergibt Länge des Variablenkopfes
E0F2-   ADC   $9B        Länge zu Zeiger auf Variablennamen addieren
E0F4-   LDY   $9C
E0F6-   BCC   $E0F9
E0F8-   INY
E0F9-   STA   $94        ergibt Zeiger hinter Variablenkopf, d.h. auf
```

```
E0FB-   STY   $95        erste Eintragung
E0FD-   RTS
---------------------
E0FE-   90 80 00 00 (20)    FP-Konst.= -32768.0005
---------------------

hole Feldindex aus Programmtext
E102-   JSR   $00B1      erstes Zeichen holen
E105-   JSR   $DD67      numerischen Ausdruck auswerten --> FAC1
E108-   LDA   $A2        Vorzeichen von FAC1
E10A-   BMI   $E119      Index ngativ --> ILLEGAL QUANTITY ERROR

FAC1 in Integer umwandeln  (Integer in A0,A1)
INT E10C-   LDA   $9D        Exponent von FAC1
E10E-   CMP   #$90       Höchstwert für gültige Integer
E110-   BCC   $E11B      Betrag von FAC1 kleiner 32768
E112-   LDA   #$FE       Zeiger auf FP-Konstante  -32768.0005
E114-   LDY   #$E0
E116-   JSR   $EBB2      FAC1 mit Konstante vergleichen
E119-   BNE   $E199      FAC1 kleiner als -32768 --> ILLEGAL QUANTITY
E11B-   JMP   $EBF2      FAC1 in Integerformat bringen
---------------------

Feldvariable suchen
E11E-   LDA   $14        Variablen-Hilfsflag
E120-   BNE   $E169      =$40 (Aufruf von STORE), keine Indizierung auswerten
E122-   LDA   $10        DIM-Flag
E124-   ORA   $12        mit Integer-Flag kombinieren
E126-   PHA              und retten
E127-   LDA   $11        String-Flag
E129-   PHA              retten
E12A-   LDY   #$00       Dimensionenzähler initialisieren
E12C-   TYA
E12D-   PHA              Anzahl auf Stack geretteter Indices
E12E-   LDA   $82        Variablenname
E130-   PHA              auf Stack retten
E131-   LDA   $81
E133-   PHA
E134-   JSR   $E102      Index aus Programmtext holen
E137-   PLA              Variablenname wieder von Stack holen
E138-   STA   $81
E13A-   PLA
E13B-   STA   $82
E13D-   PLA              Dimensionenzähler
E13E-   TAY              nach Y
E13F-   TSX
E140-   LDA   $0102,X    DIM/Integer-Flag
E143-   PHA              kopieren
E144-   LDA   $0101,X    String-Flag
E147-   PHA              kopieren
E148-   LDA   $A0        Indexbytes ersetzen Flags am alten Ort im Stack
E14A-   STA   $0102,X
E14D-   LDA   $A1
E14F-   STA   $0101,X
E152-   INY              Dimensionenzähler erhöhen
E153-   JSR   $00B7      hole nächstes Zeichen
E156-   CMP   #$2C       Komma?
E158-   BEQ   $E12C      ja, nächsten Index verarbeiten
E15A-   STY   $0F        Anzahl der Dimensionen merken
E15C-   JSR   $DEB8      prüfe ob ")" folgt, sonst SYNTAX ERROR
E15F-   PLA              Stringflag
```

```
E160-   STA   $11        wiederherstellen
E162-   PLA              Integer/DIM-Flag wiederherstellen
E163-   STA   $12
E165-   AND   #$7F
E167-   STA   $10
E169-   LDX   $6B        Anfang der Feldvariablen
E16B-   LDA   $6C
E16D-   STX   $9B        in Suchzeiger eintragen
E16F-   STA   $9C
E171-   CMP   $6E        Feldvariablen-Ende +1 erreicht?
E173-   BNE   $E179      nein
E175-   CPX   $6D        LByte vergleichen
E177-   BEQ   $E1B8      ja, Feld neu einrichten
E179-   LDY   #$00
E17B-   LDA   ($9B),Y    sonst Variablennamen vergleichen
E17D-   INY
E17E-   CMP   $81        1. Namensbyte
E180-   BNE   $E188      stimmt nicht überein
E182-   LDA   $82        2. Namensbyte
E184-   CMP   ($9B),Y    vergleichen
E186-   BEQ   $E19E      stimmt, Feldvariable gefunden, Element suchen
E188-   INY
E189-   LDA   ($9B),Y    sonst Länge der Feldvariablen
E18B-   CLC
E18C-   ADC   $9B        zum Suchzeiger addieren
E18E-   TAX
E18F-   INY
E190-   LDA   ($9B),Y
E192-   ADC   $9C
E194-   BCC   $E16D      neue Feldvariable vergleichen
--------------------
E196-   LDX   #$6B       Kode für BAD SUBSCRIPT ERROR
E198-   BIT   $35A2      --> E199-  LDX #$35 (Kode für ILLEGAL QUANTITY ERROR)
E19B-   JMP   $D412      Fehlermeldung bearbeiten
--------------------
E19E-   LDX   #$78       Kode für REDIM'D ARRAY ERROR
E1A0-   LDA   $10        DIM-Flag
E1A2-   BNE   $E19B      gesetzt --> REDIM'D ARRAY ERROR
E1A4-   LDA   $14        = $40 falls von STORE aufgerufen, sonst 00
E1A6-   BEQ   $E1AA      nicht von STORE, Feldelement suchen
E1A8-   SEC              von STORE gerufen, fertig
E1A9-   RTS
E1AA-   JSR   $E0ED      Zeiger 94,95 auf 1. Feldelement setzen
E1AD-   LDA   $0F        Anzahl angegebener Dimensionen
E1AF-   LDY   #$04
E1B1-   CMP   ($9B),Y    mit Dimensionenzahl des Feldes vergleichen
E1B3-   BNE   $E196      stimmt nicht, -->  BAD SUBSCRIPT ERROR
E1B5-   JMP   $E24B      alles ok, Feldelement suchen
--------------------

Feldvariable neu einrichten
E1B8-   LDA   $14        = $40 falls von STORE gerufen, sonst 00
E1BA-   BEQ   $E1C1      nicht von STORE , ok.
E1BC-   LDX   #$2A       Kode für OUT OF DATA ERROR
E1BE-   JMP   $D412      Fehlermeldung ausgeben
E1C1-   JSR   $E0ED      Zeiger 94,95 auf erstes Feldelement setzen
E1C4-   JSR   $D3E3      genug Platz für Variablenkopf?

Länge eines Feldelements bestimmen
E1C7-   LDA   #$00
E1C9-   TAY
```

```
E1CA-  STA  $AE        Länge, HByte
E1CC-  LDX  #$05
E1CE-  LDA  $81        1. Namensbyte  (Bit7=1 falls Integervariable)
E1D0-  STA  ($9B),Y    in Variablenkopf eintragen
E1D2-  BPL  $E1D5      keine Integervariable, da Bit7=0
E1D4-  DEX
E1D5-  INY
E1D6-  LDA  $82        2. Namensbyte  (Bit7=0 falls FP-Variable)
E1D8-  STA  ($9B),Y    in Variablenkopf eintragen
E1DA-  BPL  $E1DE      FP-Variable, da Bit7=0
E1DC-  DEX
E1DD-  DEX
E1DE-  STX  $AD        Länge also 2/3/5 Byte bei Integer/String/FP-Variable

einzelne Dimensionierungen in Var.-Kopf eintragen
E1E0-  LDA  $0F        Anzahl der Dimensionen
E1E2-  INY
E1E3-  INY
E1E4-  INY
E1E5-  STA  ($9B),Y    in Variablenkopf eintragen
E1E7-  LDX  #$0B       A,X mit 11 vorbesetzen als Default-Wert
E1E9-  LDA  #$00
E1EB-  BIT  $10        Aufruf durch DIM?
E1ED-  BVC  $E1F7      nein
E1EF-  PLA             ja, DIM-Wert vom Stack holen
E1F0-  CLC
E1F1-  ADC  #$01       um 1 erhöhen wegen 0. Element
E1F3-  TAX
E1F4-  PLA
E1F5-  ADC  #$00       HByte korrigieren falls nötig
E1F7-  INY
E1F8-  STA  ($9B),Y    DIM-Wert in Variablenkopf eintragen
E1FA-  INY
E1FB-  TXA
E1FC-  STA  ($9B),Y
E1FE-  JSR  $E2AD      neue Länge = alte Länge x Dimensionierung
E201-  STX  $AD        neue Länge merken
E203-  STA  $AE
E205-  LDY  $5E        geretteten Y-Wert holen (vgl $E2AD)
E207-  DEC  $0F        Dimensionen-Zähler dekrementieren
E209-  BNE  $E1E7      weitere Dimension eintragen

prüfen ob Platz reicht, Feld nullsetzen
E20B-  ADC  $95        Feldlänge ohne Variablenkopf (X,A) zu Zeiger auf
E20D-  BCS  $E26C      erstes Feldelement (94,95) addieren,
E20F-  STA  $95        OUT OF MEMORY ERROR falls Summe > 65535
E211-  TAY
E212-  TXA             LByte berücksichtigen
E213-  ADC  $94
E215-  BCC  $E21A
E217-  INY             Summe (neues Feldvariablen-Ende +1) in A,Y
E218-  BEQ  $E26C      OUT OF MEMORY ERROR
E21A-  JSR  $D3E3      prüfe ob genügend Platz für neues Feld
E21D-  STA  $6D        ok, neues Feldvariablen-Ende +1 eintragen
E21F-  STY  $6E
E221-  LDA  #$00
E223-  INC  $AE
E225-  LDY  $AD        als Index in Schleife
E227-  BEQ  $E22E
E229-  DEY             Schleife füllt Feld mit Nullen
E22A-  STA  ($94),Y    null eintragen
```

```
E22C-  BNE  $E229      Y nicht null, weiter
E22E-  DEC  $95        sonst HBytes dekrementieren
E230-  DEC  $AE
E232-  BNE  $E229      noch nicht fertig
E234-  INC  $95        Zeiger auf 1. Feldelement korrigieren
E236-  SEC
E237-  LDA  $6D        Feldende +1
E239-  SBC  $9B        minus Kopfanfang ergibt gesamte Länge
E23B-  LDY  #$02
E23D-  STA  ($9B),Y    Feldlänge LByte eintragen
E23F-  LDA  $6E
E241-  INY
E242-  SBC  $9C
E244-  STA  ($9B),Y    Feldlänge HByte eintragen
E246-  LDA  $10        Aufruf durch DIM?
E248-  BNE  $E2AC      ja, fertig
E24A-  INY

Feldelement suchen
E24B-  LDA  ($9B),Y    Anzahl der Dimensionen
E24D-  STA  $0F        merken
E24F-  LDA  #$00
E251-  STA  $AD        Register für laufende Nummer des Elements löschen
E253-  STA  $AE
E255-  INY
E256-  PLA             Index vom Stack nach A0,A1
E257-  TAX
E258-  STA  $A0
E25A-  PLA
E25B-  STA  $A1
E25D-  CMP  ($9B),Y    Index mit DIM-Wert vergleichen
E25F-  BCC  $E26F      HByte kleiner, ok.
E261-  BNE  $E269      Index zu groß --> BAD SUBSCRIPT ERROR
E263-  INY             HByte gleich, LByte vergleichen
E264-  TXA
E265-  CMP  ($9B),Y
E267-  BCC  $E270      kleiner, ok.
E269-  JMP  $E196      sonst --> BAD SUBSCRIPT ERROR
-------------------
E26C-  JMP  $D410      --> OUT OF MEMORY ERROR
-------------------
E26F-  INY
E270-  LDA  $AE
E272-  ORA  $AD
E274-  CLC
E275-  BEQ  $E281      beide Registerbytes null, nicht multiplizieren
E277-  JSR  $E2AD      X,A = AD,AE x DIM-Wert
E27A-  TXA
E27B-  ADC  $A0        plus Index (A0,A1)
E27D-  TAX             ergibt aktuelle Element-Position
E27E-  TYA
E27F-  LDY  $5E        hole gerettetes Y
E281-  ADC  $A1
E283-  STX  $AD        aktuelle Element-Position in Register merken
E285-  DEC  $0F        Dimensionen-Zähler dekrementieren
E287-  BNE  $E253      weitere Dimension berücksichtigen
E289-  STA  $AE        AD,AE enthält nun Element-Position im Feld
E28B-  LDX  #$05       Elementlänge aus Variablenart bestimmen
E28D-  LDA  $81        1. Namensbyte  (Bit7=1 falls Integer-Variable)
E28F-  BPL  $E292      nicht Integer
E291-  DEX
```

```
E292-   LDA   $82        2. Namensbyte (Bit7=0 falls FP-Variable)
E294-   BPL   $E298      FP-Variable
E296-   DEX
E297-   DEX
E298-   STX   $64        Länge = 2/3/5 bei Integer/String/FP-Variable
E29A-   LDA   #$00       Länge, HByte
E29C-   JSR   $E2B6      Abstand X,Y = Länge x Position
E29F-   TXA              Abstand
E2A0-   ADC   $94        zu Zeiger auf 1. Feldelement addieren
E2A2-   STA   $83
E2A4-   TYA
E2A5-   ADC   $95
E2A7-   STA   $84        83,84 zeigt nun auf gesuchtes Element
E2A9-   TAY
E2AA-   LDA   $83        A,Y ebenfalls
E2AC-   RTS
---------------------
```

Multiplikationsroutine
Faktoren in AD/AE und (9B),Y / (9B),Y+1
Produkt in X,Y bzw. X,A

```
E2AD-   STY   $5E        rette Y
E2AF-   LDA   ($9B),Y    Multiplikand  nach 64,65 übertragen
E2B1-   STA   $64
E2B3-   DEY
E2B4-   LDA   ($9B),Y
E2B6-   STA   $65
E2B8-   LDA   #$10       d.h. 16
E2BA-   STA   $99        als Bitzähler für 2Byte-Multiplikation
E2BC-   LDX   #$00       Ergebnisregister löschen
E2BE-   LDY   #$00
E2C0-   TXA              Ergebnisregister linksverschieben
E2C1-   ASL
E2C2-   TAX
E2C3-   TYA
E2C4-   ROL
E2C5-   TAY
E2C6-   BCS   $E26C      Überlauf --> OUT OF MEMORY ERROR
E2C8-   ASL   $AD
E2CA-   ROL   $AE        Multiplikator-Bit --> C
E2CC-   BCC   $E2D9      Bit = 0, nicht addieren
E2CE-   CLC              Multiplikand zu Ergebnisregister addieren
E2CF-   TXA
E2D0-   ADC   $64
E2D2-   TAX
E2D3-   TYA
E2D4-   ADC   $65
E2D6-   TAY
E2D7-   BCS   $E26C      Überlauf --> OUT OF MEMORY ERROR
E2D9-   DEC   $99        Bitzähler dekrementieren
E2DB-   BNE   $E2C0      weiter mit nächster Stelle
E2DD-   RTS              fertig
---------------------
```

Applesoft-Routine FRE

```
E2DE-   LDA   $11        String-Flag gesetzt?
E2E0-   BEQ   $E2E5      nein
E2E2-   JSR   $E600      sonst unnützen String entfernen, falls erforderlich
E2E5-   JSR   $E484      Garbage Collection durchführen
E2E8-   SEC
E2E9-   LDA   $6F        Anfang des Stringbereichs
E2EB-   SBC   $6D        minus Variablen-Ende+1
```

```
E2ED-   TAY                 ergibt freien Speicherplatz
E2EE-   LDA   $70
E2F0-   SBC   $6E

Integer in Y,A nach FAC1 übertragen
E2F2-   LDX   #$00
E2F4-   STX   $11           String-Flag löschen
E2F6-   STA   $9E           Integer in FAC1-Mantisse eintragen
E2F8-   STY   $9F
E2FA-   LDX   #$90          Exponentenwert für FAC1
E2FC-   JMP   $EB9B         weiter wie bei SGN-Routine
--------------------

Applesoft-Routine  POS
E2FF-   LDY   $24           Cursorspalte
E301-   LDA   #$00          0 als HByte
E303-   SEC
E304-   BEQ   $E2F2         nach FAC1 übertragen
--------------------

prüfe ob unzulässiger Direktmodus
E306-   LDX   $76           = $FF bei Direktmodus
E308-   INX
E309-   BNE   $E2AC (RTS)   ok, Programm-Modus, fertig
E30B-   LDX   #$95          Kode für ILLEGAL DIRECT
E30D-   BIT   $E0A2         --> E30E-  LDX #$E0 , d.h. Kode für UNDEF'D FUNCTION
E310-   JMP   $D412         zur Fehlerausgaberoutine
--------------------

Applesoft-Routine  DEF (Funktion definieren)
z.B. DEF FN F(X), wobei  F = Name, X = Argument
E313-   JSR   $E341         Namen der Funktion bestimmen
E316-   JSR   $E306         prüfe ob ILLEGAL DIRECT
E319-   JSR   $DEBB         prüfe ob "(" folgt
E31C-   LDA   #$80          sperre Integer-variablen als Argument
E31E-   STA   $14
E320-   JSR   $DFE3         suche Argument-Variable
E323-   JSR   $DD6A         muß numerisch sein
E326-   JSR   $DEB8         prüfe ob ")" folgt
E329-   LDA   #$D0          Token für "="
E32B-   JSR   $DEC0         muß folgen, sonst SYNTAX ERROR
E32E-   PHA                 1. Zeichen der FN-Definition auf Stack
E32F-   LDA   $84           Zeiger auf Argument-Variable
E331-   PHA                 auf Stack retten
E332-   LDA   $83
E334-   PHA
E335-   LDA   $B9           CHRGET-Zeiger (zeigt auf FN-Definition)
E337-   PHA                 auf Stack retten
E338-   LDA   $B8
E33A-   PHA
E33B-   JSR   $D995         CHRGET-Zeiger auf nächstes Trennzeichen setzen
E33E-   JMP   $E3AF         5 Byte von Stack in FN-Namensvariable eintragen

FN-Namensvariable aus Programmtext holen
E341-   LDA   #$C2          Token für FN
E343-   JSR   $DEC0         muß folgen, sonst SYNTAX ERROR
E346-   ORA   #$80          sperre Integervariable
E348-   STA   $14
E34A-   JSR   $DFEA         suche FN-Namensvariable
E34D-   STA   $8A           Zeiger auf FN-Namensvariable
E34F-   STY   $8B
```

```
E351-   JMP   $DD6A      prüfe ob numerische Variable (also FP), fertig

Funktion, z.B. FN F(X) auswerten
E354-   JSR   $E341      FN-Namensvariable suchen
E357-   LDA   $8B        Zeiger auf FN-Namensvariable auf Stack retten
E359-   PHA
E35A-   LDA   $8A
E35C-   PHA
E35D-   JSR   $DEB2      Klammerausdruck mit Argument auswerten --> FAC1
E360-   JSR   $DD6A      prüfe ob numerisch
E363-   PLA              Zeiger auf FN-Namensvariable wieder holen
E364-   STA   $8A
E366-   PLA
E367-   STA   $8B
E369-   LDY   #$02
E36B-   LDA   ($8A),Y    Zeiger auf Argument-Variable nach 83,84 bringen
E36D-   STA   $83
E36F-   TAX
E370-   INY
E371-   LDA   ($8A),Y
E373-   BEQ   $E30E      keine FN-Parameter vorhanden, UNDEF'D FUNCTION ERROR
E375-   STA   $84        sonst noch HByte eintragen
E377-   INY
E378-   LDA   ($83),Y    momentanen Wert der Argument-Variable
E37A-   PHA              auf Stack retten
E37B-   DEY
E37C-   BPL   $E378
E37E-   LDY   $84        X,Y zeigt auf Argument-Variable
E380-   JSR   $EB2B      FN-Argument in Argument-Variable eintragen
E383-   LDA   $B9        CHRGET-Zeiger auf Programmzeile
E385-   PHA              auf Stack retten
E386-   LDA   $B8
E388-   PHA
E389-   LDA   ($8A),Y    CHRGET-Zeiger auf FN-Ausdruck setzen
E38B-   STA   $B8
E38D-   INY
E38E-   LDA   ($8A),Y
E390-   STA   $B9
E392-   LDA   $84        Zeiger auf Argument-Variable auf Stack retten
E394-   PHA
E395-   LDA   $83
E397-   PHA
E398-   JSR   $DD67      FN-Ausdruck auswerten, Ergebnis nach FAC1
E39B-   PLA              Zeiger auf Argument-Variable nach 8A,8B
E39C-   STA   $8A        (ab E3AF wird ursprünglicher Wert eingetragen)
E39E-   PLA
E39F-   STA   $8B
E3A1-   JSR   $00B7      hole Zeichen  (hinter FN-Definitionsausdruck)
E3A4-   BEQ   $E3A9      Trennzeichen, ok. CHRGET u. Argument-Var. wie zuvor
E3A6-   JMP   $DEC9      sonst SYNTAX ERROR
E3A9-   PLA              CHRGET wieder auf Programmzeile richten
E3AA-   STA   $B8
E3AC-   PLA
E3AD-   STA   $B9

5 Byte vom Stack ab Zeiger 8A,8B eintragen
E3AF-   LDY   #$00
E3B1-   PLA
E3B2-   STA   ($8A),Y
E3B4-   PLA
E3B5-   INY
```

```
E3B6-   STA   ($8A),Y
E3B8-   PLA
E3B9-   INY
E3BA-   STA   ($8A),Y
E3BC-   PLA
E3BD-   INY
E3BE-   STA   ($8A),Y
E3C0-   PLA
E3C1-   INY
E3C2-   STA   ($8A),Y
E3C4-   RTS
--------------------

Applesoft-Funktion STR$
E3C5-   JSR   $DD6A      prüfe, ob Argument numerisch
E3C8-   LDY   #$00       als Zeiger, String wird ab $00FF,Y zwischengespeichert
E3CA-   JSR   $ED36      Argument in FAC1 --> String ab $00FF
E3CD-   PLA              Rücksprung-Adresse entfernen
E3CE-   PLA
E3CF-   LDA   #$FF       Zeiger auf Stringanfang
E3D1-   LDY   #$00
E3D3-   BEQ   $E3E7      String auswerten (Deskriptor bestimmen etc.)
--------------------

Platz für String schaffen, Deskriptor bestimmen
I: Accu enthält Stringlänge
E3D5-   LDX   $A0        Zeiger auf neuesten String im Deskriptoren-Stack
E3D7-   LDY   $A1
E3D9-   STX   $8C        als Zeiger auf aktuellen Deskriptor
E3DB-   STY   $8D
E3DD-   JSR   $E452      unteres Stringende herabsetzen
E3E0-   STX   $9E        Deskriptor für neugeschaffenen Platz:
E3E2-   STY   $9F        9E,9F = Adresse des Stringanfangs
E3E4-   STA   $9D        9D    = Stringlänge
E3E6-   RTS
--------------------

Stringdeskriptor bestimmen und auf Deskript.-Stack,
String -> Stringbereich, falls aus Page 0 oder 2
I: A,Y zeigt auf Stringanfang
O: 9D,9E,9F = Deskriptor; A0,A1 zeigt auf Deskr.Stack
   AD,AE zeigt hinter String
E3E7-   LDX   #$22       Ascii für Anführungszeichen
E3E9-   STX   $0D        als Endmarken eintragen
E3EB-   STX   $0E
E3ED-   STA   $AB        Zeiger auf Stringanfang als Arbeitszeiger
E3EF-   STY   $AC
E3F1-   STA   $9E        und für Deskriptor merken
E3F3-   STY   $9F
E3F5-   LDY   #$FF       als Index für Schleife
E3F7-   INY
E3F8-   LDA   ($AB),Y    hole Zeichen
E3FA-   BEQ   $E408      Endmarke, Ende gefunden
E3FC-   CMP   $0D        Endmarke 1?
E3FE-   BEQ   $E404      ja, Ende gefunden
E400-   CMP   $0E        Endmarke 2?
E402-   BNE   $E3F7      nein, suche weiter nach Stringende
E404-   CMP   #$22       Endmarke 1/2 = Anführungszeichen?
E406-   BEQ   $E409      ja, dann C=1
E408-   CLC
E409-   STY   $9D        Stringlänge in Deskriptor eintragen
```

```
E40B-   TYA                 Länge
E40C-   ADC   $AB           zu Stringanfang addieren (plus 1 falls mit  " )
E40E-   STA   $AD           ergibt Zeiger hinter String
E410-   LDX   $AC
E412-   BCC   $E415
E414-   INX                 HByte korrigieren falls nötig
E415-   STX   $AE
E417-   LDA   $AC           Stringanfang HByte
E419-   BEQ   $E41F         String beginnt in Zeropage (vgl STR$), kopieren
E41B-   CMP   #$02          beginnt String in Page 2? (d.h. INPUT,GET)
E41D-   BNE   $E42A         nein, nicht kopieren (bei PRINT, oder aus Programm)
E41F-   TYA                 Stringlänge
E420-   JSR   $E3D5         Platz in Stringbereich schaffen, Deskr. -> 9D,9E,9F
E423-   LDX   $AB           Zeiger auf Anfang des Originalstring
E425-   LDY   $AC
E427-   JSR   $E5E2         Originalstring in Stringbereich kopieren

Deskriptor auf Deskriptoren-Stack setzen
E42A-   LDX   $52           Zeiger für neue Eintragung in Deskriptoren-Stack
E42C-   CPX   #$5E          Deskriptorenstack ($0055-$005D) voll?
E42E-   BNE   $E435         nein, neuen Deskriptor eintragen
E430-   LDX   #$BF          Kode für FORMULA TOO COMPLEX ERROR
E432-   JMP   $D412         Fehlermeldung bearbeiten
E435-   LDA   $9D           Deskriptor in Stack eintragen, X als Zeiger
E437-   STA   $00,X
E439-   LDA   $9E
E43B-   STA   $01,X
E43D-   LDA   $9F
E43F-   STA   $02,X
E441-   LDY   #$00
E443-   STX   $A0           A0,A1 zeigt auf neuesten Deskriptor im Deskr.-Stack
E445-   STY   $A1
E447-   DEY                 = $FF
E448-   STY   $11           Stringflag setzen
E44A-   STX   $53           Adresse des obersten Deskriptors im Stack
E44C-   INX
E44D-   INX
E44E-   INX                 plus 3
E44F-   STX   $52           ergibt Zeiger für nächste Eintragung
E451-   RTS
--------------------

Platz für neuen String im Stringbereich schaffen
I: Accu = Stringlänge
O: A,X,Y = neuer Deskriptor; 71,72 neue Anfangsadresse
E452-   LSR   $13           lösche Garbage-Collection-Flag
E454-   PHA                 Länge merken
E455-   EOR   #$FF
E457-   SEC
E458-   ADC   $6F           d.h. Länge wird subtrahiert von Stringbereich-Anfang
E45A-   LDY   $70
E45C-   BCS   $E45F
E45E-   DEY                 HByte korrigieren falls nötig
E45F-   CPY   $6E           mit Feldvariablen-Ende +1 vergleichen
E461-   BCC   $E474         zu wenig Platz —> Garbage Collection
E463-   BNE   $E469         Platz reicht, weiter
E465-   CMP   $6D           HBytes gleich, prüfe LByte
E467-   BCC   $E474         zu wenig Platz —> Garbage Collection
E469-   STA   $6F           neuer Stringbereich-Anfang
E46B-   STY   $70
E46D-   STA   $71           Hilfszeiger für String-Kopierroutine
```

```
E46F-   STY   $72
E471-   TAX
E472-   PLA              damit Deskriptor in A,X,Y
E473-   RTS              fertig
E474-   LDX   #$4D       Kode für OUT OF MEMORY ERROR
E476-   LDA   $13        = $80, falls Garbage Collection schon durchgeführt
E478-   BMI   $E432      Fehlermeldung bearbeiten
E47A-   JSR   $E484      sonst Garbage Collection durchführen
E47D-   LDA   #$80       und im Garbage-Colection-Flag vermerken
E47F-   STA   $13
E481-   PLA              Stringlänge holen
E482-   BNE   $E454      prüfen, ob nun genügend Platz
---------------------

Garbage Collection Routine
Garbage E484-   LDX   $73        HIMEM, d.h. Stringbereich-Ende +1
E486-   LDA   $74
E488-   STX   $6F        als Stringbereich-Anfang (damit momentan keine Strings)
E48A-   STA   $70
E48C-   LDY   #$00
E48E-   STY   $8B        als Abbruch-Flag (bleibt null, falls alle Var. fertig)
E490-   LDA   $6D        Feldvariablen-Ende +1
E492-   LDX   $6E
E494-   STA   $9B        als Zeiger auf höchsten zu berücksichtigenden String
E496-   STX   $9C        (im folgenden als Top-String bezeichnet)
E498-   LDA   #$55       Zeiger auf Beginn des Deskriptoren-Stack
E49A-   LDX   #$00
E49C-   STA   $5E        als Suchzeiger (durchläuft sämtliche Deskriptoren)
E49E-   STX   $5F

Strings aus Deskriptoren-Stack berückichtigen
E4A0-   CMP   $52        Suchzeiger hinter letzter Eintragung im Deskr.-Stack?
E4A2-   BEQ   $E4A9      ja, damit Deskriptoren-Stack ganz berücksichtigt
E4A4-   JSR   $E523      falls Top-String, in 9B,9C und 8A,8B merken
E4A7-   BEQ   $E4A0      weiter im Deskriptoren-Stack

Strings aus einfachen Variablen berücksichtigen
E4A9-   LDA   #$07       Länge einfacher Variablen = 7 Bytes
E4AB-   STA   $8F        merken
E4AD-   LDA   $69        Anfang der einfachen Variablen
E4AF-   LDX   $6A
E4B1-   STA   $5E        als Suchzeiger
E4B3-   STX   $5F
E4B5-   CPX   $6C        Anfang der Feldvariablen erreicht?
E4B7-   BNE   $E4BD      nein
E4B9-   CMP   $6B        LByte vergleichen
E4BB-   BEQ   $E4C2      einfache Variable fertig, weiter mit Feldvariablen
E4BD-   JSR   $E519      String-Parameter merken, falls neuer Top-String
E4C0-   BEQ   $E4B5      weiter mit nächster Variablen

Strings aus Feldvariablen berücksichtigen
E4C2-   STA   $94        Zeiger auf Feldvariablen-Kopf setzen
E4C4-   STX   $95
E4C6-   LDA   #$03       Länge einzelner Elemente im Stringfeld = 3 Bytes
E4C8-   STA   $8F        merken
E4CA-   LDA   $94        Zeiger auf Kopf der aktuellen Feldvariable
E4CC-   LDX   $95
E4CE-   CPX   $6E        Feldvariablen-Ende +1 erreicht?
E4D0-   BNE   $E4D9      nein, aktuelle Feldvariable bearbeiten
E4D2-   CMP   $6D        LByte vergleichen
E4D4-   BNE   $E4D9      aktuelle Feldvariable bearbeiten
```

```
E4D6-   JMP   $E562       Ende erreicht, Top-String übernehmen

einzelne Feldvariable bearbeiten
E4D9-   STA   $5E         als Suchzeiger auf einzelne Feldelemente
E4DB-   STX   $5F
E4DD-   LDY   #$00
E4DF-   LDA   ($5E),Y     1. Namensbyte
E4E1-   TAX               retten
E4E2-   INY
E4E3-   LDA   ($5E),Y     2. Namensbyte
E4E5-   PHP               Bit7 retten
E4E6-   INY
E4E7-   LDA   ($5E),Y     Länge der Feldvariablen
E4E9-   ADC   $94         zu Zeiger auf Variablen-Kopf addieren
E4EB-   STA   $94
E4ED-   INY
E4EE-   LDA   ($5E),Y     HBytes addieren
E4F0-   ADC   $95
E4F2-   STA   $95         ergibt Zeiger auf nächste Feldvariable
E4F4-   PLP               hole Bit7 aus 2. Namensbyte
E4F5-   BPL   $E4CA       =0, d.h. keine Stringvariable, gleich weiter
E4F7-   TXA               1. Namensbyte
E4F8-   BMI   $E4CA       Bit7=1, d.h. keine Stringvariable, gleich weiter
E4FA-   INY
E4FB-   LDA   ($5E),Y     Anzahl der Dimensionen
E4FD-   LDY   #$00
E4FF-   ASL               x 2
E500-   ADC   #$05        plus 5 ergibt Variablen-Kopflänge
E502-   ADC   $5E         zu Zeiger auf Kopfanfang addieren
E504-   STA   $5E         ergibt Zeiger auf erstes Element
E506-   BCC   $E50A
E508-   INC   $5F
E50A-   LDX   $5F
E50C-   CPX   $95         nächste Feldvariable erreicht (d.h. diese fertig)?
E50E-   BNE   $E514       nein
E510-   CMP   $94         LByte vergleichen
E512-   BEQ   $E4CE       weiter mit nächster Feldvariablen
E514-   JSR   $E523       Stringparameter merken, falls neuer Top-String
E517-   BEQ   $E50C       weiter mit nächstem Element

prüfe, ob neuen Top-String gefunden
E519-   LDA   ($5E),Y     1. Namensbyte
E51B-   BMI   $E552       Bit7=1, keine Stringvariable, gleich weiter
E51D-   INY
E51E-   LDA   ($5E),Y     2. Namensbyte
E520-   BPL   $E552       Bit7=0, keine Stringvariable, gleich weiter
E522-   INY
E523-   LDA   ($5E),Y     Länge des Strings
E525-   BEQ   $E552       = null, nicht Top-String
E527-   INY
E528-   LDA   ($5E),Y     Anfangsadresse des Strings
E52A-   TAX
E52B-   INY
E52C-   LDA   ($5E),Y     nach X,A
E52E-   CMP   $70         Adresse im Stringbereich?
E530-   BCC   $E538       nein, String also noch nicht berücksichtigt
E532-   BNE   $E552       ja, String bereits als aktuell erkannt
E534-   CPX   $6F         LByte vergleichen
E536-   BCS   $E552
E538-   CMP   $9C         liegt String über bisherigem Top-String?
E53A-   BCC   $E552       nein, weiter
```

```
E53C-   BNE   $E542      ja, neuer Top-String gefunden
E53E-   CPX   $9B        LByte vergleichen
E540-   BCC   $E552
E542-   STX   $9B        Zeiger auf neuen Top-String setzen
E544-   STA   $9C
E546-   LDA   $5E        Zeiger auf zugehörigen Deskriptor
E548-   LDX   $5F
E54A-   STA   $8A        merken
E54C-   STX   $8B
E54E-   LDA   $8F        Abstand zu nächstem Deskriptor
E550-   STA   $91        merken
E552-   LDA   $8F        Abstand zu nächstem Deskriptor
E554-   CLC
E555-   ADC   $5E        zu Suchzeiger addieren
E557-   STA   $5E
E559-   BCC   $E55D
E55B-   INC   $5F
E55D-   LDX   $5F        ergibt neuen Suchzeiger
E55F-   LDY   #$00
E561-   RTS              weiter mit nächstem Deskriptor

Top-String in Stringbereich übernehmen
E562-   LDX   $8B        Deskriptorzeiger, =0 falls kein Top-String gefunden
E564-   BEQ   $E55D      =0, Garbage Collection beendet
E566-   LDA   $91        = 7 bei einfacher Variabler, sonst = 3
E568-   AND   #$04       ergibt 4 oder 0
E56A-   LSR              ergibt 2 oder 0
E56B-   TAY              als Index (bei einfacher Variable Namen überspringen)
E56C-   STA   $91        merken
E56E-   LDA   ($8A),Y    Stringlänge aus Deskriptor
E570-   ADC   $9B        plus Zeiger auf String
E572-   STA   $96        ergibt Endadresse +1  (für Verschiebe-Routine)
E574-   LDA   $9C
E576-   ADC   #$00
E578-   STA   $97
E57A-   LDA   $6F        bisheriger Anfang der Strings
E57C-   LDX   $70
E57E-   STA   $94        als neues Blockende +1
E580-   STX   $95
E582-   JSR   $D39A      String in Stringbereich kopieren
E585-   LDY   $91
E587-   INY              gemerkten Index erhöhen
E588-   LDA   $94        neue Stringadresse in 94,95+1
E58A-   STA   ($8A),Y    in Deskriptor eintragen
E58C-   TAX
E58D-   INC   $95
E58F-   LDA   $95        X,A zeigt auf neuen Anfang der Strings
E591-   INY
E592-   STA   ($8A),Y    HByte noch in Deskriptor eintragen
E594-   JMP   $E488      weiter mit Suche nach neuem Top-String
---------------------

Strings verketten
E597-   LDA   $A1        Zeiger auf Deskriptor des 1. Strings
E599-   PHA              auf Stack retten
E59A-   LDA   $A0
E59C-   PHA
E59D-   JSR   $DE60      2. String aus Programmtext holen (A0,A1 Deskr.-Zeiger)
E5A0-   JSR   $DD6C      prüfe ob String
E5A3-   PLA              Deskriptorzeiger für 1. String
E5A4-   STA   $AB        nach AB,AC bringen
```

```
E5A6-   PLA
E5A7-   STA   $AC
E5A9-   LDY   #$00
E5AB-   LDA   ($AB),Y     Länge des 1. Strings
E5AD-   CLC
E5AE-   ADC   ($A0),Y     plus Länge des 2. Strings
E5B0-   BCC   $E5B7       ok, nicht zu lang
E5B2-   LDX   #$B0        Kode für STRING TOO LONG ERROR
E5B4-   JMP   $D412       Fehlermeldung bearbeiten
E5B7-   JSR   $E3D5       Platz für verketteten String schaffen
E5BA-   JSR   $E5D4       1. Teilstring an neuen Platz kopieren
E5BD-   LDA   $8C         Zeiger auf Deskriptor für 2. String im D.-Stack
E5BF-   LDY   $8D
E5C1-   JSR   $E604       Deskriptor vom Deskriptoren-Stack entfernen
E5C4-   JSR   $E5E6       2. Teilstring an neuen Platz hinter ersten kopieren
E5C7-   LDA   $AB         Zeiger auf Deskriptor für 1. String im D.-Stack
E5C9-   LDY   $AC
E5CB-   JSR   $E604       Deskriptor vom Deskriptoren-Stack entfernen
E5CE-   JSR   $E42A       Deskriptor für verketteten String auf Deskr.-Stack
E5D1-   JMP   $DD95       zurück zur Routine "Ausdruck auswerten"
--------------------

String kopieren
I: AB,AC Zeiger auf Deskriptor; 71,72 Zieladresse
O: 71,72 zeigt hinter neuen String
COPYSTR
E5D4-   LDY   #$00
E5D6-   LDA   ($AB),Y     Stringlänge retten
E5D8-   PHA
E5D9-   INY
E5DA-   LDA   ($AB),Y     Stringadresse
E5DC-   TAX               nach X,Y bringen
E5DD-   INY
E5DE-   LDA   ($AB),Y
E5E0-   TAY
E5E1-   PLA               Länge holen
E5E2-   STX   $5E         Adresse auf Zeiger übertragen
E5E4-   STY   $5F
E5E6-   TAY               Y als Schleifenzähler
E5E7-   BEQ   $E5F3       Stringlänge null
E5E9-   PHA               sonst Länge retten
E5EA-   DEY               Schleife kopiert Zeichen für Zeichen
E5EB-   LDA   ($5E),Y
E5ED-   STA   ($71),Y
E5EF-   TYA               Z-Flag setzen
E5F0-   BNE   $E5EA       weiter in Schleife
E5F2-   PLA               Länge wieder holen
E5F3-   CLC
E5F4-   ADC   $71         und zu Anfangsadresse addieren
E5F6-   STA   $71         ergibt Zeiger hinter neuen String
E5F8-   BCC   $E5FC
E5FA-   INC   $72
E5FC-   RTS
--------------------

unnütze Strings entfernen
O: Deskriptor in A,5E,5F bzw. in A,X,Y
E5FD-   JSR   $DD6C       prüfe, ob letzter Ausdruck ein String war
E600-   LDA   $A0         Zeiger auf Deskriptor im Deskriptoren-Stack
E602-   LDY   $A1
E604-   STA   $5E         auf Hilfszeiger übertragen
E606-   STY   $5F
```

```
E608-   JSR   $E635      Deskriptor vom D.-Stack entfernen, falls obenauf
E60B-   PHP              Z-Flag gesetzt, falls Deskriptor entfernt, --> retten
E60C-   LDY   #$00
E60E-   LDA   ($5E),Y    Stringdeskriptor nach A,X,Y
E610-   PHA
E611-   INY
E612-   LDA   ($5E),Y
E614-   TAX
E615-   INY
E616-   LDA   ($5E),Y
E618-   TAY
E619-   PLA              Stringlänge
E61A-   PLP              gerettetes Z-Flag holen
E61B-   BNE   $E630      Deskriptor wurde nicht entfernt, weiter
E61D-   CPY   $70        Stringadresse = Anfang des Stringbereichs?
E61F-   BNE   $E630      nein
E621-   CPX   $6F        LByte vergleichen
E623-   BNE   $E630      nein, nicht der unterste String --> nicht löschen
E625-   PHA              sonst Länge retten
E626-   CLC
E627-   ADC   $6F        Anfang des Stringbereichs um Stringlänge erhöhen
E629-   STA   $6F
E62B-   BCC   $E62F
E62D-   INC   $70
E62F-   PLA              A,X,Y = Deskriptor
E630-   STX   $5E        Adresse nach 5E,5F kopieren
E632-   STY   $5F
E634-   RTS
--------------------
```

Deskriptor vom D.-Stack entfernen, falls obenauf
I: A,Y zeigt auf Stringdeskriptor
O: Z-Flag =1, falls Deskriptor entfernt wurde

```
E635-   CPY   $54        Adresse des obersten Deskriptors auf Stack, HByte?
E637-   BNE   $E645      stimmt nicht überein, fertig
E639-   CMP   $53        LByte prüfen
E63B-   BNE   $E645      stimmt nicht überein
E63D-   STA   $52        sonst Zeiger für nächste Eintragung setzen
E63F-   SBC   #$03       minus 3
E641-   STA   $53        ergibt Zeiger auf neuen obersten Deskriptor im Stack
E643-   LDY   #$00       Z-Flag setzen
E645-   RTS
--------------------
```

Applesoft-Funktion CHR$

```
E646-   JSR   $E6FB      Ascii-Kode in FAC1 als Integer nach X
E649-   TXA              retten
E64A-   PHA
E64B-   LDA   #$01       als Stringlänge
E64D-   JSR   $E3DD      Platz im Stringbereich schaffen, Deskr.-> 9D,9E,9F
E650-   PLA              Ascii-Kode
E651-   LDY   #$00
E653-   STA   ($9E),Y    eintragen
E655-   PLA              Rücksprungadresse entfernen
E656-   PLA
E657-   JMP   $E42A      Deskriptor auf Deskriptoren-Stack
---------------------
```

Applesoft-Funktion LEFT$

```
E65A-   JSR   $E6B9      Deskriptorzeiger nach 8C,8D; Parameter nach A und X
E65D-   CMP   ($8C),Y    Parameter mit Stringlänge vergleichen
```

```
E65F-   TYA                 Accu nullsetzen

Einsprung von RIGHT$
E660-   BCC   $E666         Anfang des Teilstrings < Stringlänge, ok
E662-   LDA   ($8C),Y       sonst Stringlänge
E664-   TAX                 =   Länge des Teilstrings
E665-   TYA
E666-   PHA                 Position des Teilstrings im Originalstring auf Stack
E667-   TXA                 Länge des Teilstrings

Einsprung von MID$
E668-   PHA                 Länge des Teilstrings auf Stack
E669-   JSR   $E3DD         Platz für neuen String im Stringbereich
E66C-   LDA   $8C           Zeiger auf Stringdeskriptor
E66E-   LDY   $8D
E670-   JSR   $E604         Originalstring entfernen, falls möglich
E673-   PLA                 Länge des Teilstrings
E674-   TAY                 merken
E675-   PLA                 Position des Teilstrings im Original (0 bei LEFT$)
E676-   CLC
E677-   ADC   $5E           Adresse des Originalstrings plus Position
E679-   STA   $5E           ergibt Adresse des Teilstrings
E67B-   BCC   $E67F
E67D-   INC   $5F
E67F-   TYA                 gemerkte Länge des Teilstrings
E680-   JSR   $E5E6         Teilstring in Stringbereich übertragen
E683-   JMP   $E42A         Deskriptor auf Deskriptoren-Stack
---------------------

Applesoft-Funktion  RIGHT$
E686-   JSR   $E6B9         Deskriptorzeiger nach 8C,8D; Parameter nach A und X
E689-   CLC
E68A-   SBC   ($8C),Y       Parameter minus (Stringlänge +1), da C=0
E68C-   EOR   #$FF          negieren ergibt Länge des Teilstrings
E68E-   JMP   $E660         weiter bei LEFT$-Routine
---------------------

Applesoft-Funktion  MID$
E691-   LDA   #$FF          Default-Wert für Länge des Teilstrings
E693-   STA   $A1           merken
E695-   JSR   $00B7         hole letztes Zeichen
E698-   CMP   #$29          Ascii für ")" ?
E69A-   BEQ   $E6A2         ja, kein Längen-Parameter angegeben
E69C-   JSR   $DEBE         sonst muß Komma folgen
E69F-   JSR   $E6F8         Längen-Parameter aus Programmtext holen (nach A1)
E6A2-   JSR   $E6B9         Deskriptorzeiger nach 8C,8D; Parameter nach A und X
E6A5-   DEX                 Position des Teilstrings
E6A6-   TXA
E6A7-   PHA                 retten
E6A8-   CLC
E6A9-   LDX   #$00
E6AB-   SBC   ($8C),Y       minus (Originalstring-Länge +1)
E6AD-   BCS   $E667         Pos. außerhalb Originalstring --> X=0 (kein Teilstring)
E6AF-   EOR   #$FF          ergibt maximale Länge des Teilstrings
E6B1-   CMP   $A1           diese mit Längen-Parameter vergleichen
E6B3-   BCC   $E668         Parameter zu groß, weiter mit maximaler Länge
E6B5-   LDA   $A1           sonst Längen-Parameter holen
E6B7-   BCS   $E668         damit weiter bei LEFT$-Routine
---------------------

LEFT$/MID$/RIGHT$-Parameter vom Stack holen, vgl DF1A
```

```
E6B9-   JSR   $DEB8      prüfe ob ")" folgt
E6BC-   PLA              Rücksprungadresse nach Y,91 retten
E6BD-   TAY
E6BE-   PLA
E6BF-   STA   $91
E6C1-   PLA              Rücksprungadresse nach $DF4C entfernen
E6C2-   PLA
E6C3-   PLA              Positions-Parameter vom Stack
E6C4-   TAX              nach X
E6C5-   PLA              Zeiger auf Stringdeskriptor vom Stack
E6C6-   STA   $8C        nach 8C,8D
E6C8-   PLA
E6C9-   STA   $8D
E6CB-   LDA   $91        gemerkte Rücksprungadresse wieder auf Stack
E6CD-   PHA
E6CE-   TYA
E6CF-   PHA
E6D0-   LDY   #$00
E6D2-   TXA              A=X (Positions-Parameter)
E6D3-   BEQ   $E6F2      =0 -->  ILLEGAL QUANTITY ERROR
E6D5-   RTS
--------------------

Applesoft-Funktion  LEN
E6D6-   JSR   $E6DC      Deskr. vom Deskr.-Stack entfernen, Stringlänge nach Y
E6D9-   JMP   $E301      Integer in Y nach FAC1 bringen
--------------------

Hilfsroutine für LEN, ASC, VAL
E6DC-   JSR   $E5FD      unnützen Deskr. vom Deskr.-Stack entfernen, Länge -> A
E6DF-   LDX   #$00
E6E1-   STX   $11        String-Flag löschen
E6E3-   TAY              Stringlänge nach Y
E6E4-   RTS
--------------------

Applesoft-Funktion  ASC
E6E5-   JSR   $E6DC      Deskr. vom Deskr.-Stack entfernen, Deskr. --> A,5E,5F
E6E8-   BEQ   $E6F2      Länge = 0, --> ILLEGAL QUANTITY ERROR
E6EA-   LDY   #$00       hole erstes Stringzeichen (in Ascii)
E6EC-   LDA   ($5E),Y
E6EE-   TAY              Ascii-Kode nach Y
E6EF-   JMP   $E301      Integer in Y nach FAC1
--------------------
E6F2-   JMP   $E199      ILLEGAL QUANTITY ERROR ausgeben
--------------------

1-Byte-Integer aus Programmtext holen
I: CHRGET zeigt auf numerischen Ausdruck
O: Integer in X bzw. A0,A1
Getint  E6F5-   JSR   $00B1      neues Zeichen holen
E6F8-   JSR   $DD67      numerischen Ausdruck auswerten  (nach FAC1)
E6FB-   JSR   $E108      FAC1 in Integer umwandeln
E6FE-   LDX   $A0        Integerzahl, HByte
E700-   BNE   $E6F2      Integer > 255, --> ILLEGAL QUANTITY ERROR
E702-   LDX   $A1        LByte nach X
E704-   JMP   $00B7      kehre zurück mit neuem Zeichen im Accu
--------------------

Applesoft-Funktion  VAL
E707-   JSR   $E6DC      Deskr. vom Deskr.-Stack entfernen, Deskr. --> A,5E,5F
```

```
E70A-   BNE   $E70F      Stringlänge nicht null
E70C-   JMP   $E84E      sonst FAC1 nullsetzen, fertig
E70F-   LDX   $B8        CHRGET-Zeiger
E711-   LDY   $B9
E713-   STX   $AD        in AD,AE retten
E715-   STY   $AE
E717-   LDX   $5E        Stringadresse
E719-   STX   $B8        nach CHRGET-Zeiger übertragen
E71B-   CLC
E71C-   ADC   $5E        Stringlänge plus Stringadresse
E71E-   STA   $60        ergibt Zeiger hinter String
E720-   LDX   $5F        dasselbe mit HBytes
E722-   STX   $B9
E724-   BCC   $E727
E726-   INX
E727-   STX   $61
E729-   LDY   #$00
E72B-   LDA   ($60),Y    erstes Zeichen hinter String
E72D-   PHA              auf Stack retten
E72E-   LDA   #$00
E730-   STA   ($60),Y    durch null ersetzen (als Endmarke)
E732-   JSR   $00B7      hole erstes Stringzeichen
E735-   JSR   $BC4A      String in FP-Konstante (in FAC1) umwandeln
E738-   PLA              erstes Zeichen hinter String
E739-   LDY   #$00
E73B-   STA   ($60),Y    wiederherstellen
E73D-   LDX   $AD        gemerkten CHRGET-Zeiger
E73F-   LDY   $AE
E741-   STX   $B8        wiederherstellen
E743-   STY   $B9
E745-   RTS
--------------------

hole 2-Byte-Integer und 1-Byte Integer aus Progr.Text
O: Integers in 50,51 (2-Byte) und A0,A1 bzw X (1-Byte)
E746-   JSR   $DD67      numerischen Ausdruck auswerten
E749-   JSR   $E752      FAC1 nach Integer in 50,51
E74C-   JSR   $DEBE      prüfe, ob Komma folgt
E74F-   JMP   $E6F8      1-Byte-Integer nach A0,A1 bzw. X bringen
--------------------

FAC1 in Integer umwandeln (in 50,51)
E752-   LDA   $9D        Exponent von FAC1
E754-   CMP   #$91       zu groß?
E756-   BCS   $E6F2      ja, Betrag von FAC1 > 65535 --> ILLEGAL QUANTITY ERROR
E758-   JSR   $EBF2      FAC1 in Integerformat bringen (Integer in A0,A1)
E75B-   LDA   $A0        Integer
E75D-   LDY   $A1
E75F-   STY   $50        in Integer-Register übertragen
E761-   STA   $51
E763-   RTS
--------------------

Applesoft-Funktion  PEEK
E764-   LDA   $50        Integer-Register auf Stack retten
E766-   PHA
E767-   LDA   $51
E769-   PHA
E76A-   JSR   $E752      Adresse (in FAC1) in Integer umwandeln (in 50,51)
E76D-   LDY   #$00
E76F-   LDA   ($50),Y    Inhalt der Speicherzelle holen
```

```
E771-   TAY                 nach Y retten
E772-   PLA                 Integer-Register wiederherstellen
E773-   STA   $51
E775-   PLA
E776-   STA   $50
E778-   JMP   $E301         Integer in Y nach FAC1 bringen
---------------------

Applesoft-Routine  POKE
E77B-   JSR   $E746         POKE-Adresse nach 50,51; neuen Wert nach X bringen
E77E-   TXA                 neuen Inhalt der Speicherzelle
E77F-   LDY   #$00
E781-   STA   ($50),Y       eintragen
E783-   RTS
---------------------

Applesoft-Routine  WAIT
E784-   JSR   $E746         Adresse nach 50,51; Prüf-Bitmaske nach X bringen
E787-   STX   $85           Maske merken
E789-   LDX   #$00          Default-Wert für Sollwert-Maske
E78B-   JSR   $00B7         hole Zeichen aus Programmtext
E78E-   BEQ   $E793         Trennzeichen, keine Sollwert-Maske angegeben
E790-   JSR   $E74C         sonst Sollwert-Maske holen
E793-   STX   $86           Sollwert-Maske merken
E795-   LDY   #$00          Beginn der Warteschleife
E797-   LDA   ($50),Y       Inhalt der WAIT-Adresse holen
E799-   EOR   $86           Sollwert-Maske invertiert Bits, die 0 sein sollen
E79B-   AND   $85           Prüf-Bitmaske holt relevante Bits heraus
E79D-   BEQ   $E797         kein relevantes Bit auf Sollwert, warten
E79F-   RTS
---------------------
ARITHMETIK-ROUTINEN

FAC1 = FAC1 + 0.5
E7A0-   LDA   #$64          Zeiger auf FP-Konstante 0.5
E7A2-   LDY   #$EE
E7A4-   JMP   $E7BE         FAC1 = FAC1 + (A,Y)
---------------------

FAC1 = (A,Y) - FAC1
E7A7-   JSR   $E9E3         (A,Y) nach FAC2 übertragen
E7AA-   LDA   $A2           Vorzeichen von FAC1  Einsprung Fac 1=Fac2-Fac1
E7AC-   EOR   #$FF          invertieren
E7AE-   STA   $A2           damit FAC1 mit umgekehrtem Vorzeichen
E7B0-   EOR   $AA           Vorzeichen FAC1 mit Vorzeichen FAC2 verknüpfen
E7B2-   STA   $AB           als Kombi-Vorzeichen merken
E7B4-   LDA   $9D           Exponent von FAC1 holen
E7B6-   JMP   $E7C1         FAC1 = (A,Y) + (-FAC1)
---------------------

Hilfsroutine für Additionsroutine
E7B9-   JSR   $E8F0         Stellenwertigkeiten von FAC1, FAC2 angleichen
E7BC-   BCC   $E7FA         zurück (immer)
---------------------

ADDAY   FAC1 = (A,Y) + FAC1
E7BE-   JSR   $E9E3         (A,Y) nach FAC2 übertragen
E7C1-   BNE   $E7C6         FAC1 ungleich null, Addition durchführen
E7C3-   JMP   $EB53         sonst FAC1 gleich FAC2 setzen, fertig
---------------------

FAC1 = FAC1 + FAC2
```

Fac1 = Fac1 + Fac2

```
E7C6-   LDX   $AC        Rundungsstelle von FAC1
E7C8-   STX   $92        merken
E7CA-   LDX   #$A5       Offsetzeiger auf FAC2  (für Mantissenverschiebung)
E7CC-   LDA   $A5        Exponent von FAC2
E7CE-   TAY              in Y merken
E7CF-   BEQ   $E79F (RTS) FAC2 = 0, damit fertig
E7D1-   SEC
E7D2-   SBC   $9D        minus Exponent von FAC1
E7D4-   BEQ   $E7FA      beide gleich, keine Mantissenangleichung notwendig
E7D6-   BCC   $E7EA      Exp 1  >  Exp 2
E7D8-   STY   $9D        Exp 2 größer, als Exponent des Ergebnisses eintragen
E7DA-   LDY   $AA        Vorzeichen von FAC2
E7DC-   STY   $A2        als Vorzeichen des Ergebnisses, da Betrag FAC1 größer
E7DE-   EOR   #$FF       Differenz der Exponenten negieren
E7E0-   ADC   #$00       damit Differenz sicher negativ
E7E2-   LDY   #$00
E7E4-   STY   $92        gemerkte Rundungsstelle von FAC1 löschen
E7E6-   LDX   #$9D       Offsetzeiger auf FAC1  (für Mantissenverschiebung)
E7E8-   BNE   $E7EE      immer springen
E7EA-   LDY   #$00
E7EC-   STY   $AC        Rundungsstelle von FAC1 löschen

Mantissen angleichen
E7EE-   CMP   #$F9       Exponentendifferenz kleiner als -7 ?
E7F0-   BMI   $E7B9      ja, kleinere Zahl nach rechts schieben bis Mantisse ok
E7F2-   TAY              sonst kleinere Zahl Bitweise verschieben bis ok
E7F3-   LDA   $AC        Rundungsstelle
E7F5-   LSR   $01,X      höchstes Mantissenbyte rechtsverschieben
806F E7F7-   JSR   $E907 817F weiter bei Verschieberoutine

Vorzeichenanalyse
8072 E7FA-   BIT   $AB        Bit7 = 0 falls die Summanden gleiches Vorzeichen haben
E7FC-   BPL   $E855      Vorzeichen gleich --> Mantissen addieren
E7FE-   LDY   #$9D       Offsetzeiger auf FAC1
E800-   CPX   #$A5       war Exp2 kleiner als Exp1?
E802-   BEQ   $E806      ja
E804-   LDY   #$A5
E806-   SEC              damit Y Zeiger für Zahl mit größerem Exp, X für andere

Mantissen subtrahieren, Differenz nach FAC1
E807-   EOR   #$FF       Rundungsstellen subtrahieren
E809-   ADC   $92
E80B-   STA   $AC
E80D-   LDA   $0004,Y    M1 (höchstwertiges Mantissenbyte)
E810-   SBC   $04,X
E812-   STA   $A1
E814-   LDA   $0003,Y    M2
E817-   SBC   $03,X
E819-   STA   $A0
E81B-   LDA   $0002,Y    M3
E81E-   SBC   $02,X
E820-   STA   $9F
E822-   LDA   $0001,Y    M4 subtrahieren
E825-   SBC   $01,X
E827-   STA   $9E
E829-   BCS   $E82E      Mantissendifferenz positiv, ok
E82B-   JSR   $E89E      FAC1 komplementieren, da negativ

FAC1 normieren (Mantisse linksbündig machen)
E82E-   LDY   #$00
E830-   TYA              Accu als Zähler (für Exponentenkorrektur benötigt)
```

```
E831-   CLC
E832-   LDX   $9E        höchstes Mantissenbyte
E834-   BNE   $E880      nicht leer, --> zur Verschiebung um einzelne Bits
E836-   LDX   $9F        sonst um ganzes Byte verschieben:
E838-   STX   $9E        M2 --> M1
E83A-   LDX   $A0
E83C-   STX   $9F        M3 --> M2
E83E-   LDX   $A1
E840-   STX   $A0        M4 --> M3
E842-   LDX   $AC
E844-   STX   $A1        Rundungsstelle  --> M4
E846-   STY   $AC        null --> Rundungsstelle
E848-   ADC   #$08       Zähler um 8 erhöhen
E84A-   CMP   #$20       bereits um 4 Bytes verschoben?
E84C-   BNE   $E832      nein, weiter
E84E-   LDA   #$00       ja, also ist Mantisse null

FAC1 nullsetzen
E850-   STA   $9D        Exponent und Vorzeichen nullsetzen
E852-   STA   $A2
E854-   RTS
---------------------

Mantissen addieren, Summe nach FAC1
E855-   ADC   $92        Rundungsstelle addieren
E857-   STA   $AC
E859-   LDA   $A1        M1 (höchstwertiges Mantissenbyte) addieren
E85B-   ADC   $A9
E85D-   STA   $A1
E85F-   LDA   $A0        M2
E861-   ADC   $A8
E863-   STA   $A0
E865-   LDA   $9F        M3
E867-   ADC   $A7
E869-   STA   $9F
E86B-   LDA   $9E        M4
E86D-   ADC   $A6
E86F-   STA   $9E
E871-   JMP   $E88D      Überlaufstelle einfügen, falls nötig
---------------------

FAC1 normieren (linksbündig machen, Bit-Weise)
E874-   ADC   #$01       Zähler erhöhen
E876-   ASL   $AC        Rundungsstelle verschieben
E878-   ROL   $A1        M4 verschieben
E87A-   ROL   $A0        M3
E87C-   ROL   $9F        M2
E87E-   ROL   $9E        M1
E880-   BPL   $E874      höchstwertigstes Bit =0, weiterschieben
E882-   SBC              Accu = Anzahl der Bits, um die linksverschoben wurde
E883-   SBC   $9D        minus Exponent
E885-   BCS   $E84E      Exponent zu klein, FAC1 nullsetzen
E887-   EOR   #$FF       Vorzeichen des Exponenten korrigieren
E889-   ADC   #$01
E88B-   STA   $9D        und eintragen  (C sicher null)
E88D-   BCC   $E89D      kein Überlauf, fertig
E88F-   INC   $9D        Exponent erhöhen
E891-   BEQ   $E8D5      Exponentenüberlauf, --> OVERFLOW ERROR
E893-   ROR   $9E        Mantisse 1 Bit rechtsverschieben, C=1 ins höchste Bit
E895-   ROR   $9F
E897-   ROR   $A0
```

```
E899-   ROR   $A1
E89B-   ROR   $AC
E89D-   RTS
--------------------

FAC1 komplementieren
E89E-   LDA   $A2        Vorzeichen umdrehen
E8A0-   EOR   #$FF
E8A2-   STA   $A2
E8A4-   LDA   $9E        Mantisse invertieren
E8A6-   EOR   #$FF
E8A8-   STA   $9E
E8AA-   LDA   $9F
E8AC-   EOR   #$FF
E8AE-   STA   $9F
E8B0-   LDA   $A0
E8B2-   EOR   #$FF
E8B4-   STA   $A0
E8B6-   LDA   $A1
E8B8-   EOR   #$FF
E8BA-   STA   $A1
E8BC-   LDA   $AC
E8BE-   EOR   #$FF
E8C0-   STA   $AC

FAC1 um 1 an geringwertigster Bitstelle erhöhen
E8C2-   INC   $AC        Rundungsstelle um 1 Bit erhöhen
E8C4-   BNE   $E8D4      kein Überlauf ins nächste Byte
E8C6-   INC   $A1        sonst M4 erhöhen usw.
E8C8-   BNE   $E8D4
E8CA-   INC   $A0
E8CC-   BNE   $E8D4
E8CE-   INC   $9F
E8D0-   BNE   $E8D4
E8D2-   INC   $9E
E8D4-   RTS
--------------------
E8D5-   LDX   #$45       Kode für OVERFLOW ERROR
E8D7-   JMP   $D412      Fehlermeldung bearbeiten
--------------------

Hilfsregister rechtsverschieben
E8DA-   LDX   #$61

Mantisse rechtsverschieben
I: Accu = negative Anzahl zu verschiebender Bits
   X = Zeiger auf Mantisse (in Zeropage)
E8DC-   LDY   $04,X      Mantisse um ganze Bytes verschieben:
E8DE-   STY   $AC        M4 --> Rundungsstelle
E8E0-   LDY   $03,X
E8E2-   STY   $04,X      M3 --> M4
E8E4-   LDY   $02,X
E8E6-   STY   $03,X      M2 --> M3
E8E8-   LDY   $01,X
E8EA-   STY   $02,X      M1 --> M2
E8EC-   LDY   $A4        =00 falls positiv, =255 falls negativ
E8EE-   STY   $01,X      --> M1

allgemeiner Einsprung
E8F0-   ADC   #$08       Zähler erhöhen
E8F2-   BMI   $E8DC      noch kleiner null, also um ganzes Byte verschieben
```

```
E8F4-   BEQ   $E8DC      gleich null, um ganzes Byte verschieben
E8F6-   SBC   #$08       sonst Zähler wieder auf alten Wert bringen
E8F8-   TAY              von nun ab zählt Y die Verschiebungen um einzelne Bits
E8F9-   LDA   $AC        Rundungsstelle
E8FB-   BCS   $E911      Zähler ist null, keine weiteren Verschiebungen

Verschiebung um einzelne Bits
E8FD-   ASL   $01,X      höchstes Bit nach C
E8FF-   BCC   $E903      =0, bei Rechtsverschiebung vorne 0-Bit anfügen
E901-   INC   $01,X      sonst Bit 1 setzen
E903-   ROR   $01,X      ASL rückgängig machen, vorne anzufügendes Bit in C
E905-   ROR   $01,X      M1 rechtsverschieben, C nach Bit 7
E907-   ROR   $02,X      M2 verschieben
E909-   ROR   $03,X      M3
E90B-   ROR   $04,X      M4
E90D-   ROR              Rundungsstelle verschieben
E90E-   INY              Zähler erhöhen
E90F-   BNE   $E8FD      weiter schieben
E911-   CLC
E912-   RTS
---------------------
E913-   81 00 00 00 00     FP-Konstante  1
---------------------

Reihenkoeffizienten für LOG-Routine
E918-   03                 Grad des Polynoms
E919-   7F 5E 56 CB 79     0.434255942
E91E-   80 13 9B 0B 64     0.576584541
E923-   80 76 38 93 16     0.961800759
E928-   82 38 AA 3B 20     2.88539007
---------------------
E92D-   80 35 04 F3 34     FP-Konstante  SQR(0.5)
---------------------
E932-   81 35 04 F3 34     FP-Konstante  SQR(2)
---------------------
E937-   80 80 00 00 00     FP-Konstante  - 0.5
---------------------
E93C-   80 31 72 17 F8     FP-Konstante  ln 2
---------------------

Applesoft-Funktion  LOG
nach der Formel ln(x * 2^k) = ln(x) + k * ln(2)
E941-   JSR   $EB82      Vorzeichen von FAC1 testen
E944-   BEQ   $E948      FAC1 = 0 —> ILLEGAL QUANTITY ERROR
E946-   BPL   $E94B      FAC1 > 0  ok, natürlichen Logarithmus bestimmen
E948-   JMP   $E199      sonst ILLEGAL QUANTITY ERROR
E94B-   LDA   $9D        Exponent des Arguments
E94D-   SBC   #$7F       minus 128 ergibt wahren Exponenten
E94F-   PHA              retten
E950-   LDA   #$80       als neuer Exponent, damit  0.5 <= FAC1 < 1
E952-   STA   $9D        in Exponentenbyte eintragen
E954-   LDA   #$2D       Zeiger auf FP-Konstante SQR(0.5)
E956-   LDY   #$E9
E958-   JSR   $E7BE      FAC1 = FAC1 + SQR(0.5)
E95B-   LDA   #$32       Zeiger auf FP-Konstante SQR(2)
E95D-   LDY   #$E9
E95F-   JSR   $EA66      FAC1 = SQR(2) / FAC1
E962-   LDA   #$13       Zeiger auf FP-Konstante  1
E964-   LDY   #$E9
E966-   JSR   $E7A7      FAC1 = 1 - FAC1, damit x' = (x-k) / (x+k); k=SQR(0.5)
E969-   LDA   #$18       Zeiger auf Polynom-Koeffizienten
```

FAEC

```
E96B-   LDY   #$E9
E96D-   JSR   $EF5C     Polynom (x') auswerten, ergibt ln(x) / ln(2) + 0.5
E970-   LDA   #$37      Zeiger auf FP-Konstante  -0.5
E972-   LDY   #$E9
E974-   JSR   $E7BE     dies zu FAC1 addieren ergibt  ln(x) / ln(2)
E977-   PLA             Exponent zur Basis 2  ( d.h. k ) holen
E978-   JSR   $ECD5     zu gerundetem FAC1 addieren ergibt  ln(x) / ln(2) + k
E97B-   LDA   #$3C      Zeiger auf FP-Konstante  ln(2)
E97D-   LDY   #$E9      (nach Multiplikation ergibt sich  ln(x) + k * ln(2) )

FAC1 = FAC1 * (A,Y)
E97F-   JSR   $E9E3     (A,Y) nach FAC2 bringen
E982-   BNE   $E987     FAC1 ungleich 0, Multiplikation ausführen
E984-   JMP   $E9E2     FAC1 = 0, also auch Produkt = 0, fertig
--------------------

FAC1 = FAC1 * FAC2
E987-   JSR   $EA0E     Exponenten von FAC1 und FAC2 addieren
E98A-   LDA   #$00
E98C-   STA   $62       Hilfsregister löschen
E98E-   STA   $63
E990-   STA   $64
E992-   STA   $65
E994-   LDA   $AC       FAC1 Byte für Byte mit FAC2 multiplizieren, Ergebnis
E996-   JSR   $E9B0     im Hilfsregister aufsummieren
E999-   LDA   $A1       M4
E99B-   JSR   $E9B0     mit FAC2 multiplizieren etc.
E99E-   LDA   $A0       M3
E9A0-   JSR   $E9B0
E9A3-   LDA   $9F       M2
E9A5-   JSR   $E9B0
E9A8-   LDA   $9E       M1
E9AA-   JSR   $E9B5
E9AD-   JMP   $EAE6     Hilfsregister nach FAC1 bringen und linksbündig machen
--------------------

Accu * FAC2 zu Hilfsregister addieren und verschieben
E9B0-   BNE   $E9B5     Accu ungleich 0, --> multiplizieren
E9B2-   JMP   $E8DA     sonst Hilfsregister um 1 Byte nach rechts verschieben
E9B5-   LSR             Bit 0 nach C bringen
E9B6-   ORA   #$80      Bit 7 setzen  (als Abbruchmarke)
E9B8-   TAY             merken
E9B9-   BCC   $E9D4     Bit = 0, nur schieben
E9BB-   CLC             Bit = 1, addieren und dann schieben
E9BC-   LDA   $65       FAC2 zu Hilfsregister addieren
E9BE-   ADC   $A9
E9C0-   STA   $65       M4
E9C2-   LDA   $64
E9C4-   ADC   $A8
E9C6-   STA   $64       M3
E9C8-   LDA   $63
E9CA-   ADC   $A7
E9CC-   STA   $63       M2
E9CE-   LDA   $62
E9D0-   ADC   $A6
E9D2-   STA   $62       M1
E9D4-   ROR   $62       Hilfsregister um 1 Bit nach rechts schieben
E9D6-   ROR   $63
E9D8-   ROR   $64
E9DA-   ROR   $65
E9DC-   ROR   $AC       letztes Bit in Rundungsstelle schieben
```

```
E9DE-   TYA                 Multiplikator
E9DF-   LSR                 nächstes Bit nach C bringen
E9E0-   BNE   $E9B8         Endmarke noch nicht erreicht, neues Bit bewerten
E9E2-   RTS                 fertig
--------------------

(A,Y) nach FAC2 bringen
E9E3-   STA   $5E           Zeiger übernehmen
E9E5-   STY   $5F
E9E7-   LDY   #$04          Index auf 4. Mantissenbyte
E9E9-   LDA   ($5E),Y       M4
E9EB-   STA   $A9
E9ED-   DEY
E9EE-   LDA   ($5E),Y       M3
E9F0-   STA   $A8
E9F2-   DEY
E9F3-   LDA   ($5E),Y       M2
E9F5-   STA   $A7
E9F7-   DEY
E9F8-   LDA   ($5E),Y       M1 enthält Vorzeichen in Bit 7
E9FA-   STA   $AA           als Vorzeichen von FAC2 eintragen
E9FC-   EOR   $A2           mit Vorzeichen von FAC1 verknüpfen
E9FE-   STA   $AB           Kombi-Vorzeichen eintragen
EA00-   LDA   $AA           M1
EA02-   ORA   #$80          Bit 7 setzen
EA04-   STA   $A6           in FAC2 eintragen
EA06-   DEY
EA07-   LDA   ($5E),Y       Exponent übertragen
EA09-   STA   $A5
EA0B-   LDA   $9D           mit Exponent von FAC1 zurück
EA0D-   RTS
--------------------

Exponenten von FAC1 und FAC2 addieren
EA0E-   LDA   $A5           Exp 2
EA10-   BEQ   $EA31         FAC2 = 0, also auch das Produkt, fertig
EA12-   CLC
EA13-   ADC   $9D           Exp 1  addieren
EA15-   BCC   $EA1B         Produkt kleiner als 1
EA17-   BMI   $EA36         Produkt zu groß, OVERFLOW ERROR
EA19-   CLC
EA1A-   BIT   $1410         --> EA1B-  BPL $EA31  d.h. Produkt nullsetzen
EA1D-   ADC   #$80          Exponent des Ergebnisses in Normalform
EA1F-   STA   $9D           nach FAC1 bringen
EA21-   BNE   $EA26         Produkt ungleich 0
EA23-   JMP   $E852         sonst noch Vorzeichen nullsetzen, fertig
EA26-   LDA   $AB           Kombi-Vorzeichen
EA28-   STA   $A2           als Vorzeichen des Ergebnisses
EA2A-   RTS
--------------------

für EXP-Routine Argument prüfen
EA2B-   LDA   $A2           Vorzeichen von FAC1 (Argument der EXP-Funktion)
EA2D-   EOR   #$FF          negieren
EA2F-   BMI   $EA36         FAC1 positiv, d.h. Argument zu groß --> OVERFLOW ERROR
EA31-   PLA                 FAC1 negativ, d.h. Argument zu klein --> nullsetzen
EA32-   PLA                 Rücksprungadresse entfernen
EA33-   JMP   $E84E         FAC1 nullsetzen, fertig
--------------------
EA36-   JMP   $E8D5         OVERFLOW ERROR anzeigen
--------------------

FAC1 = 10 * FAC1
```

Fac1 = 10 × Fac 1

```
EA39-   JSR   $EB63      FAC1 nach FAC2 übertragen
EA3C-   TAX              Exponent von FAC1
EA3D-   BEQ   $EA4F      FAC1 = 0, also auch das Produkt, fertig
EA3F-   CLC
EA40-   ADC   #$02       plus 2 im Exponenten ergibt den 4-fachen Wert
EA42-   BCS   $EA36      Overflow
EA44-   LDX   #$00
EA46-   STX   $AB        Kombi-Vorzeichen plus
EA48-   JSR   $E7CE      FAC1 = FAC1 + FAC2, das ergibt den 5-fachen Wert
EA4B-   INC   $9D        plus 1 im Exponenten (d.h. verdoppeln) --> 10-fach
EA4D-   BEQ   $EA36      Overflow
EA4F-   RTS
--------------------
EA50-   84 20 00 00 00   FP-Konstante  10
--------------------

FAC1 = FAC1 / 10
EA55-   JSR   $EB63      FAC1 nach FAC2 bringen
EA58-   LDA   #$50       Zeiger auf FP-Konstante 10
EA5A-   LDY   #$EA
EA5C-   LDX   #$00
EA5E-   STX   $AB        Kombi-Vorzeichen plus
EA60-   JSR   $EAF9      Konstante nach FAC1 übertragen
EA63-   JMP   $EA69      Division durchführen
--------------------

FAC1 = (A,Y) / FAC1
EA66-   JSR   $E9E3      (A,Y) nach FAC2 übertragen

FAC1 = FAC2 / FAC1
EA69-   BEQ   $EAE1      FAC1 = 0, DIVISION BY ZERO ERROR
EA6B-   JSR   $EB72      FAC1 runden
EA6E-   LDA   #$00
EA70-   SEC
EA71-   SBC   $9D        ergibt negativen Exponenten
EA73-   STA   $9D
EA75-   JSR   $EA0E      Exponenten und Vorzeichen des Ergebisses bestimmen
EA78-   INC   $9D
EA7A-   BEQ   $EA36      Quotient zu groß, OVERFLOW ERROR
EA7C-   LDX   #$FC       als Index in Zeropage (überschreitet Pagegrenze nicht!)
EA7E-   LDA   #$01       setze Bit0 = 1 als Abbruchmarke
EA80-   LDY   $A6        vergleiche FAC2 mit FAC1
EA82-   CPY   $9E        M1
EA84-   BNE   $EA96
EA86-   LDY   $A7        M2 vergleichen, da M1 gleich etc.
EA88-   CPY   $9F
EA8A-   BNE   $EA96
EA8C-   LDY   $A8
EA8E-   CPY   $A0
EA90-   BNE   $EA96
EA92-   LDY   $A9
EA94-   CPY   $A1
EA96-   PHP              rette Statusflags
EA97-   ROL              C --> Bit0; Bit7 --> C
EA98-   BCC   $EAA3      Abbruchmarke noch nicht erreicht
EA9A-   INX              neues Byte des Quotienten voll,
EA9B-   STA   $65,X      abspeichern in Hilfsregister (62-65)
EA9D-   BEQ   $EAD1      X=0, d.h. 4 Byte im Ergebnis, jetzt noch 2 weitere Bit
EA9F-   BPL   $EAD5      X=1, Division fertig, Routine abschließen
EAA1-   LDA   #$01       Abbruch-Bit für nächstes Ergebnisbyte
EAA3-   PLP              Statusflags holen
```

```
EAA4-   BCS   $EAB4    FAC1 kleiner FAC2, —> subtrahieren
EAA6-   ASL   $A9      FAC2 um 1 Bit nach links schieben
EAA8-   ROL   $A8
EAAA-   ROL   $A7
EAAC-   ROL   $A6
EAAE-   BCS   $EA96    FAC2 nun sicher größer als FAC1, Vergleich nicht nötig
EAB0-   BMI   $EA80    höchstes Bit in FAC2 =1, Vergleich durchführen
EAB2-   BPL   $EA96    FAC2 sicher kleiner als FAC1, Vergleich nicht nötig

Mantissen subtrahieren
EAB4-   TAY            Ergebnisbyte retten
EAB5-   LDA   $A9      M4
EAB7-   SBC   $A1      subtrahieren
EAB9-   STA   $A9
EABB-   LDA   $A8      M3
EABD-   SBC   $A0
EABF-   STA   $A8
EAC1-   LDA   $A7      M2
EAC3-   SBC   $9F
EAC5-   STA   $A7
EAC7-   LDA   $A6      M1
EAC9-   SBC   $9E
EACB-   STA   $A6
EACD-   TYA            Ergebnisbyte holen
EACE-   JMP   $EAA6    weiter in Hauptroutine
-----------------------
EAD1-   LDA   #$40     Abbruchmarke für 2 weitere Ergebnis-Bits
EAD3-   BNE   $EAA3    Division abschließen
-----------------------
EAD5-   ASL            im letzten Ergebnisbyte nur Bits 0 und 1 gültig
EAD6-   ASL
EAD7-   ASL
EAD8-   ASL
EAD9-   ASL
EADA-   ASL            diese damit in Bits 6 und 7 (d.h. linksbündig)
EADB-   STA   $AC      in Rundungsstelle eintragen
EADD-   PLP            Stack aktualisieren
EADE-   JMP   $EAE6    Hilfsregister nach FAC1 und linksbündig machen
-----------------------
EAE1-   LDX   #$85     Kode für DIVISION BY ZERO ERROR
EAE3-   JMP   $D412    Fehlermeldung bearbeiten
-----------------------

Hilfsregister linksbündig nach FAC1 übertragen
EAE6-   LDA   $62      M1
EAE8-   STA   $9E
EAEA-   LDA   $63      M2
EAEC-   STA   $9F
EAEE-   LDA   $64      M3
EAF0-   STA   $A0
EAF2-   LDA   $65      M4
EAF4-   STA   $A1
EAF6-   JMP   $E82E    FAC1 linksbündig machen
-----------------------

(A,Y) nach FAC1 übertragen
EAF9-   STA   $5E      Zeiger übernehmen
EAFB-   STY   $5F
EAFD-   LDY   #$04     Index auf geringwertigstes Mantissenbyte
EAFF-   LDA   ($5E),Y  M4
EB01-   STA   $A1      übertragen
```

AYTOC1

```
EB03-   DEY
EB04-   LDA   ($5E),Y    M3
EB06-   STA   $A0
EB08-   DEY
EB09-   LDA   ($5E),Y    M2
EB0B-   STA   $9F
EB0D-   DEY
EB0E-   LDA   ($5E),Y    höchstes Mantissenbyte enthält Vorzeichen in Bit7
EB10-   STA   $A2        als Vorzeichen von FAC1
EB12-   ORA   #$80       Bit7 =1 setzen
EB14-   STA   $9E        M1 eintragen
EB16-   DEY
EB17-   LDA   ($5E),Y    Exponent
EB19-   STA   $9D        nach FAC1 übertragen
EB1B-   STY   $AC        Rundungsstelle löschen
EB1D-   RTS
--------------------

FAC1 nach FAC4, FAC3, (85,86) oder (X,Y) übertragen
EB1E-   LDX   #$98       Einsprung für FAC4, X als Zeiger
EB20-   BIT   $93A2      --> EB21-  LDX #$93  Einsprung für FAC3, X als Zeiger
EB23-   LDY   #$00       HByte
EB25-   BEQ   $EB2B      immer springen
EB27-   LDX   $85        Einsprung für (85,86)
EB29-   LDY   $86

FAC1 nach (X,Y) übertragen
EB2B-   JSR   $EB72      FAC1 runden
EB2E-   STX   $5E        Zeiger auf Zieladresse übernehmen
EB30-   STY   $5F
EB32-   LDY   #$04       als Index
EB34-   LDA   $A1        M4
EB36-   STA   ($5E),Y    übertragen
EB38-   DEY
EB39-   LDA   $A0        M3
EB3B-   STA   ($5E),Y
EB3D-   DEY
EB3E-   LDA   $9F        M2
EB40-   STA   ($5E),Y
EB42-   DEY
EB43-   LDA   $A2        Vorzeichen von FAC1
EB45-   ORA   #$7F
EB47-   AND   $9E        in höchstes Mantissenbyte einbauen
EB49-   STA   ($5E),Y    und übertragen
EB4B-   DEY
EB4C-   LDA   $9D        Exponent
EB4E-   STA   ($5E),Y    übertragen
EB50-   STY   $AC        Rundungsstelle löschen
EB52-   RTS
--------------------

FAC2 nach FAC1 übertragen
EB53-   LDA   $AA        Vorzeichen von FAC2
EB55-   STA   $A2        --> Vorzeichen von FAC1
EB57-   LDX   #$05       Index für Schleife
EB59-   LDA   $A4,X      Byte für Byte übertragen
EB5B-   STA   $9C,X
EB5D-   DEX
EB5E-   BNE   $EB59      weiter
EB60-   STX   $AC        Rundungsstelle löschen
EB62-   RTS
```

Fac 1 Fac 4 ?
Fac 1 Fac 3
Fac 1 X Y

```
---------------------

FAC1 gerundet nach FAC2 übertragen
EB63-   JSR   $EB72      FAC1 runden
EB66-   LDX   #$06       Index für Schleife (4Byte-Mantisse, Exp, Vorzeichen)
EB68-   LDA   $9C,X      Byte für Byte übertragen
EB6A-   STA   $A4,X
EB6C-   DEX
EB6D-   BNE   $EB68      weiter
EB6F-   STX   $AC        Rundungsstelle löschen
EB71-   RTS
---------------------

FAC1 runden
EB72-   LDA   $9D        Exponent von FAC1
EB74-   BEQ   $EB71      FAC1 = 0, fertig
EB76-   ASL   $AC        Bit7 der Rundungsstelle --> C
EB78-   BCC   $EB71      abrunden, d.h. fertig
EB7A-   JSR   $E8C6      aufrunden, Mantisse um 1 Bit erhöhen
EB7D-   BNE   $EB71      Mantisse nicht null, d.h. kein Überlauf-Bit
EB7F-   JMP   $E88F      sonst Überlauf-Bit berücksichtigen
---------------------

Vorzeichen von FAC1 testen
EB82-   LDA   $9D        Exponent von FAC1
EB84-   BEQ   $EB8F      FAC1 = 0
EB86-   LDA   $A2        Vorzeichen von FAC1
EB88-   ROL              Bit7 --> C                Accu  C  Z  N   falls
EB89-   LDA   #$FF       d.h. -1                   ----------------------
EB8B-   BCS   $EB8F      falls negativ             255   1  0  1   FAC1 < 0
EB8D-   LDA   #$01       sonst 1                   0     ?  1  0   FAC1 = 0
EB8F-   RTS                                        1     0  0  0   FAC1 > 0
---------------------

Applesoft-Funktion  SGN
EB90-   JSR   $EB82      Vorzeichen von FAC1 testen          +   0   -
EB93-   STA   $9E        Kode nach M1                        1   0   255
EB95-   LDA   #$00
EB97-   STA   $9F        M2 = 0
EB99-   LDX   #$88       als Exponent für Integer
EB9B-   LDA   $9E        Vorzeichen
EB9D-   EOR   #$FF       umkehren
EB9F-   ROL              Bit7 --> C        damit C-Flag      1   1   0
EBA0-   LDA   #$00
EBA2-   STA   $A1        M4
EBA4-   STA   $A0        und M3 löschen
EBA6-   STX   $9D        Exponent eintragen
EBA8-   STA   $AC        Rundungsstelle löschen
EBAA-   STA   $A2        Vorzeichen löschen
EBAC-   JMP   $E829      invertieren falls negativ, linksbündig machen
---------------------

Applesoft-Funktion  ABS
EBAF-   LSR   $A2        Bit7 nullsetzen, d.h. positives Vorzeichen
EBB1-   RTS              fertig
---------------------

FAC1 mit (A,Y) vergleichen
EBB2-   STA   $60        Zeiger übernehmen
EBB4-   STY   $61
EBB6-   LDY   #$00
```

```
EBB8-   LDA   ($60),Y    Exponent des Vergleichs-Registers
EBBA-   INY
EBBB-   TAX              in X merken
EBBC-   BEQ   $EB82      Vergleichs-Register = 0, --> Vorzeichen FAC1 testen
EBBE-   LDA   ($60),Y    höchstes Mantissenbyte, Vorzeichen in Bit7
EBC0-   EOR   $A2        mit Vorzeichen von FAC1 vergleichen
EBC2-   BMI   $EB86      verschiedene Vorzeichen, --> Vorzeichen FAC1 testen
EBC4-   CPX   $9D        Exponenten vergleichen ( C=0 falls FAC1 größer)
EBC6-   BNE   $EBE9      ungleich
EBC8-   LDA   ($60),Y    sonst Mantissen vergleichen
EBCA-   ORA   #$80       Bit7 setzen (statt Vorzeichen-Bit)
EBCC-   CMP   $9E        höchste Mantissenbytes vergleichen
EBCE-   BNE   $EBE9      ungleich
EBD0-   INY
EBD1-   LDA   ($60),Y    M2 vergleichen, da M1 gleich etc.
EBD3-   CMP   $9F
EBD5-   BNE   $EBE9
EBD7-   INY
EBD8-   LDA   ($60),Y    M3 vergleichen
EBDA-   CMP   $A0
EBDC-   BNE   $EBE9
EBDE-   INY
EBDF-   LDA   #$7F
EBE1-   CMP   $AC        Rundungsstelle mit #$7F vergleichen, C setzen
EBE3-   LDA   ($60),Y    M4 vergleichen, C je nach Rundungsstelle
EBE5-   SBC   $A1
EBE7-   BEQ   $EC11      FAC1 = Vergleichsregister, fertig
EBE9-   LDA   $A2        Vorzeichen von FAC1
EBEB-   BCC   $EBEF      FAC1 betragsmäßig größer, --> Vergleich auswerten
EBED-   EOR   #$FF       sonst Vorzeichen umkehren für Vergleichsauswertung
EBEF-   JMP   $EB88      Vergleichs-Flags setzen wie bei Vorzeichentest
---------------------

FAC1 in Integerformat bringen (Integer in A0,A1)
EBF2-   LDA   $9D        Exponent von FAC1
EBF4-   BEQ   $EC40      FAC1 = 0, Mantisse mit Nullen füllen
EBF6-   SEC
EBF7-   SBC   #$A0       ergibt Stellenwert für geringwertigstes Bit
EBF9-   BIT   $A2        Vorzeichen von FAC1 prüfen
EBFB-   BPL   $EC06      positiv
EBFD-   TAX              sonst Stellenwertigkeit retten
EBFE-   LDA   #$FF       zum Auffüllen bei Verschieben (vgl. $E8EC)
EC00-   STA   $A4
EC02-   JSR   $E8A4      Mantisse invertieren
EC05-   TXA              Stellenwertigkeit wieder holen
EC06-   LDX   #$9D       Zeiger auf FAC1 für Verschiebe-Routine
EC08-   CMP   #$F9       Stellenwertigkeit mit -7 vergleichen
EC0A-   BPL   $EC12      größer als -8
EC0C-   JSR   $E8F0      Mantisse verschieben, bis Stellenwertigkeit ok
EC0F-   STY   $A4        "Auffüllbyte" nullsetzen
EC11-   RTS              fertig
EC12-   TAY              Stellenwertigkeit für Verschiebung nach Y
EC13-   LDA   $A2        Vorzeichen (im Bit7)
EC15-   AND   #$80       herausholen
EC17-   LSR   $9E        höchstes Mantissen-Byte nach rechts schieben
EC19-   ORA   $9E        Vorzeichen-Bit eintragen
EC1B-   STA   $9E
EC1D-   JSR   $E907      weiter bei Verschiebe-Routine
EC20-   STY   $A4        "Auffüllbyte" nullsetzen
EC22-   RTS
---------------------
```

Applesoft-Funktion INT

```
EC23-   LDA   $9D        Exponent von FAC1
EC25-   CMP   #$A0       schon richtige Stellenwetigkeit für Integerformat?
EC27-   BCS   $EC49      FAC1 schon ganzzahlig, fertig
EC29-   JSR   $EBF2      sonst FAC1 in Integerformat bringen
EC2C-   STY   $AC        Rundungsstelle löschen
EC2E-   LDA   $A2        Vorzeichen von FAC1 nach Accu
EC30-   STY   $A2        Vorzeichenbyte löschen
EC32-   EOR   #$80       Vorzeichen in Bit7 negieren
EC34-   ROL              Damit C=1 falls positives Argument
EC35-   LDA   #$A0       Exponent für Integerformat
EC37-   STA   $9D        eintragen
EC39-   LDA   $A1        niedrigstes Mantissenbyte
EC3B-   STA   $0D        retten (vgl. $EEB8, $EF2A)
EC3D-   JMP   $E829      FAC1 linksbündig machen
--------------------
EC40-   STA   $9E        Mantisse von FAC1 mit Wert in Accu füllen (vgl. $EBF4)
EC42-   STA   $9F
EC44-   STA   $A0
EC46-   STA   $A1
EC48-   TAY
EC49-   RTS
--------------------
```

Zahlenstring in FP-Konstante (in FAC1) umwandeln
I: CHRGET zeigt auf Stringadresse, Accu = 1.Zeichen

```
EC4A-   LDY   #$00
EC4C-   LDX   #$0A       als Zählindex
EC4E-   STY   $99,X      Zeropage-Adressen $99 - $A3 löschen
EC50-   DEX
EC51-   BPL   $EC4E      nächstes Byte löschen
```

Auswertung der Stringmantisse

```
EC53-   BCC   $EC64      1. Stringzeichen ist Ziffer, zur Mantissenauswertung
EC55-   CMP   #$2D       Ascii für "-" ?
EC57-   BNE   $EC5D      nein, weiter
EC59-   STX   $A3        Bit7 = 1, d.h. Mantissenvorzeichen = minus
EC5B-   BEQ   $EC61      weiter mit nächstem Zeichen (immer springen)
EC5D-   CMP   #$2B       Ascii für "+" ?
EC5F-   BNE   $EC66      nein, weiter
EC61-   JSR   $00B1      neues Stringzeichen holen
EC64-   BCC   $EC01      Ziffer, zur Mantissenauswertung
EC66-   CMP   #$2E       Ascii für "." (d.h. Dezimalpunkt) ?
EC68-   BEQ   $EC98      Ja, Dezimalpunkt-Flag setzen
```

Auswertung des Stringexponenten

```
EC6A-   CMP   #$45       Ascii für "E" ?
EC6C-   BNE   $EC9E      nein, keine Zehnerpotenz angegeben, Routine abschließen
EC6E-   JSR   $00B1      ja, neues Zeichen holen
EC71-   BCC   $EC8A      Ziffer, zur Exponenten-Auswertung
EC73-   CMP   #$C9       Token für "-" ?
EC75-   BEQ   $EC85      ja, Stringexponent negativ
EC77-   CMP   #$2D       Ascii für "-" ?
EC79-   BEQ   $EC85      ja, Stringexponent negativ
EC7B-   CMP   #$C8       Token für "+" ?
EC7D-   BEQ   $EC87      ja, ignorieren
EC7F-   CMP   #$2B       Ascii für "+" ?
EC81-   BEQ   $EC87      ja, ignorieren
EC83-   BNE   $EC8C      sonst String fertig ausgewertet, Routine abschließen
EC85-   ROR   $9C        C=1 --> Bit7, d.h. negatives Exponenten-Vorzeichen
EC87-   JSR   $00B1      neues Stringzeichen holen
```

```
EC8A-   BCC   $ECE8    Ziffer, zur Exponenten-Auswertung

Dezimalstellen auswerten
EC8C-   BIT   $9C      Vorzeichen des Exponenten prüfen
EC8E-   BPL   $EC9E    positiv, Stellenwertigkeit berücksichtigen
EC90-   LDA   #$00     negativ, Exponentenregister negativ machen
EC92-   SEC
EC93-   SBC   $9A
EC95-   JMP   $ECA0    Stellenwertigkeit berücksichtigen
--------------------
EC98-   ROR   $9B      C=1 --> Bit7, d.h. Dezimalpunkt-Flag setzen
EC9A-   BIT   $9B      teste Bit6 ( =1, falls Bit7 vorher schon gesetzt war)
EC9C-   BVC   $EC61    Bit6=0, weiter mit nächstem Zeichen
--------------------
EC9E-   LDA   $9A      Exponentenregister (enthält Stellenwertigkeit)
ECA0-   SEC
ECA1-   SBC   $99      minus Anzahl der Nachkommastellen
ECA3-   STA   $9A      ergibt wahre Stellenwertigkeit
ECA5-   BEQ   $ECB9    =0, keine dezimale Verschiebung ( denn 10^0 = 1)
ECA7-   BPL   $ECB2    positiv, FAC1 mit entspr. Zehnerpotenz multiplizieren

durch Zehnerpotenz dividieren
ECA9-   JSR   $EA55    FAC1 = FAC1 / 10
ECAC-   INC   $9A      als Zähler für Zehnerpotenz
ECAE-   BNE   $ECA9    weiter durch 10 dividieren
ECB0-   BEQ   $ECB9    Mantissen-Vorzeichen beachten, dann fertig

mit Zehnerpotenz multiplizieren
ECB2-   JSR   $EA39    FAC1 = FAC1 * 10
ECB5-   DEC   $9A      als Zähler für Zehnerpotenz
ECB7-   BNE   $ECB2    weiter mit 10 multiplizieren
ECB9-   LDA   $A3      Vorzeichen der Mantisse
ECBB-   BMI   $ECBE    negativ, Vorzeichenwechsel
ECBD-   RTS            positiv, fertig
ECBE-   JMP   $EED0    Vorzeichenwechsel bei FAC1 durchführen, dann fertig

Mantissenziffer auswerten
ECC1-   PHA            Ziffer retten
ECC2-   BIT   $9B      teste Dezimalpunkt-Flag
ECC4-   BPL   $ECC8    Vorkommastelle
ECC6-   INC   $99      Zähler für Nachkommastellen erhöhen
ECC8-   JSR   $EA39    FAC1 = FAC1 * 10, d.h. um eine Dezimale verschieben
ECCB-   PLA            Ziffer (in Ascii) holen
ECCC-   SEC
ECCD-   SBC   #$30     ergibt reine Ziffer
ECCF-   JSR   $ECD5    diese zu FAC1 addieren
ECD2-   JMP   $EC61    weiter mit neuem Zeichen

Zahl in Accu zu FAC1 addieren
ECD5-   PHA            Zahl retten
ECD6-   JSR   $EB63    FAC1 gerundet nach FAC2 übertragen
ECD9-   PLA            Zahl wieder holen
ECDA-   JSR   $EB93    und in FP-Konstante (in FAC1) umwandeln
ECDD-   LDA   $AA      Vorzeichen von FAC1
ECDF-   EOR   $A2      mit Vorzeichen von FAC2 verknüpfen
ECE1-   STA   $AB      ergibt Kombi-Vorzeichen
ECE3-   LDX   $9D      Exponent von FAC1 nach Accu,
ECE5-   JMP   $E7C1    Addition durchführen

Exponentenziffer auswerten
ECE8-   LDA   $9A      bisheriger Exponent
```

```
ECEA-   CMP   #$0A        mit 10 vergleichen
ECEC-   BCC   $ECF7       kleiner, Ziffer in Exponent eintragen
ECEE-   LDA   #$64        sonst 100 als neuer Exponentenbetrag
ECF0-   BIT   $9C         teste Exponenten-Vorzeichen
ECF2-   BMI   $ED05       negativ, damit wird FAC1 später zu null
ECF4-   JMP   $E8D5       positiv, OVERFLOW ERROR
ECF7-   ASL               bisheriger Exponent verdoppeln
ECF8-   ASL               damit 4-fach
ECF9-   CLC
ECFA-   ADC   $9A         einmal addiert ergibt 5-facher Wert
ECFC-   ASL               verdoppeln, ergibt 10-fachen Wert
ECFD-   CLC
ECFE-   LDY   #$00
ED00-   ADC   ($B8),Y     Ascii-Kode der neuen Ziffer addieren
ED02-   SEC
ED03-   SBC   #$30        Ergebnis korrigieren
ED05-   STA   $9A         neue Zehnerpotenz eintragen
ED07-   JMP   $EC87       weiter mit neuem Zeichen
---------------------
ED0A-   9B 3E BC 1F FD     FP-Konst.   99 999 999.906
ED0F-   9E 6E 6B 27 FD     FP-Konst.  999 999 999.25
ED14-   9E 6E 6B 28 00     FP-Konst. 1000 000 000
---------------------
```

ERRLNR

```
Ausgabe der Zeilennummer bei Fehlermeldungen
ED19-   LDA   #$58        Zeiger auf String  " IN "
ED1B-   LDY   #$D3
ED1D-   JSR   $ED31       String drucken
ED20-   LDA   $76         aktuelle Zeilennummer (dort trat der Fehler auf)
ED22-   LDX   $75
ED24-   STA   $9E         nach FAC1 übertragen
ED26-   STX   $9F
ED28-   LDX   #$90        als Exponent
ED2A-   SEC
ED2B-   JSR   $EBA0       FAC1 normieren
ED2E-   JSR   $ED34       FAC1 in Zahlenstring umwandeln
ED31-   JMP   $DB3A       String drucken
---------------------

Umwandlung von FAC1 in Zahlenstring
O: Stringadresse $0100
```

Str out 1

```
ED34-   LDY   #$01        Offset zu $00FF für Zeiger auf neuen String
```

Str out

```
Einsprung von STR$ mit Y=0 (d.h. String ab $00FF)
ED36-   LDA   #$2D        Ascii für "-"
ED38-   DEY
ED39-   BIT   $A2         Vorzeichen von FAC1
ED3B-   BPL   $ED41       positiv, weiter
ED3D-   INY
ED3E-   STA   $00FF,Y     sonst Minuszeichen eintragen
ED41-   STA   $A2         Bit7 nullsetzen, d.h. Betrag von FAC1 bilden
ED43-   STY   $AD         Zeiger auf neuen String retten
ED45-   INY
ED46-   LDA   #$30        Ascii für "0"
ED48-   LDX   $9D         Exponent von FAC1
ED4A-   BNE   $ED4F       FAC1 ungleich 0
ED4C-   JMP   $EE57       sonst Ascii-Kode für "0" eintragen und fertig

FAC1 in den Bereich (10^8 , 10^9) bringen
ED4F-   LDA   #$00
ED51-   CPX   #$80        Exponent von FAC1 mit #$80 vergleichen
```

```
ED53-   BEQ   $ED57      Betrag von FAC1 zwischen 0.5 und 1
ED55-   BCS   $ED60      Betrag von FAC1 >= 1
ED57-   LDA   #$14       Zeiger auf FP-Konstante  1000 000 000
ED59-   LDY   #$ED
ED5B-   JSR   $E97F      FAC1 = FAC1 * 1000 000 000
ED5E-   LDA   #$F7       -9
ED60-   STA   $99        als Stellenwert-Flag eintragen
ED62-   LDA   #$0F       Zeiger auf FP-Konstante  999 999 999.25
ED64-   LDY   #$ED
ED66-   JSR   $EBB2      vergleiche FAC1 mit  999 999 999.25
ED69-   BEQ   $ED89      gleich, ok, im Sollbereich
ED6B-   BPL   $ED7F      zu groß, durch 10 dividieren
ED6D-   LDA   #$0A       Zeiger auf FP-Konstante  99 999 999.906
ED6F-   LDY   #$ED
ED71-   JSR   $EBB2      vergleiche FAC1 mit  99 999 999.906
ED74-   BEQ   $ED78      gleich, zu klein, mit 10 multiplizieren
ED76-   BPL   $ED86      größer, ok, im Sollbereich
-----------------------
ED78-   JSR   $EA39      kleiner, FAC1 mit 10 multiplizieren
ED7B-   DEC   $99        Stellenwert-Flag verringern
ED7D-   BNE   $ED6D      prüfen, ob nun im Sollbereich
-----------------------
ED7F-   JSR   $EA55      FAC1 durch 10 dividieren
ED82-   INC   $99        Stellenwert-Flag erhöhen
ED84-   BNE   $ED62      prüfen, ob nun im Sollbereich

Stringformat bestimmen (Exponentialdarstellung?)
ED86-   JSR   $E7A0      FAC1 = FAC1 + 0.5 (d.h. aufrunden)
ED89-   JSR   $EBF2      FAC1 in Integerformat bringen (keine Nachkommastellen)
ED8C-   LDX   #$01
ED8E-   LDA   $99        Stellenwert-Flag
ED90-   CLC
ED91-   ADC   #$0A       plus 10 ergibt wahren Exponenten +2
ED93-   BMI   $ED9E      <0, d.h. wahrer Exp < -2, d.h. Betrag  < 0.01
ED95-   CMP   #$0B       d.h. wahren Exponenten mit 9 vergleichen
ED97-   BCS   $ED9F      Betrag >= 10^9, deswegen Exponentialdarstellung
ED99-   ADC   #$FF       minus 1
ED9B-   TAX              wahrer Exponent +1 in X merken
ED9C-   LDA   #$02
ED9E-   SEC
ED9F-   SBC   #$02                 normale Darst.        Exponential-Darst.
EDA1-   STA   $9A        ergibt    0                     Exponenten
EDA3-   STX   $99        ergibt    Exponenten +1         1
EDA5-   TXA              Anzahl signifikanter Vorkommastellen
EDA6-   BEQ   $EDAA      =0,  d.h.  0.1  <=  Betrag  <  1
EDA8-   BPL   $EDBD      >0   d.h.           Betrag  >  1
EDAA-   LDY   $AD        Zeiger auf den String
EDAC-   LDA   #$2E       Ascii für Dezimalpunkt
EDAE-   INY
EDAF-   STA   $00FF,Y    in String eintragen
EDB2-   TXA              Anzahl signifikanter Vorkommastellen
EDB3-   BEQ   $EDBB      0.1  <=  Betrag  <  1
EDB5-   LDA   #$30       0.01 <=  Betrag < 0.1, daher "0" hinter Dezimalpunkt
EDB7-   INY
EDB8-   STA   $00FF,Y    Ascii für "0" eintragen
EDBB-   STY   $AD        Zeiger auf den String wieder retten

Approximation der einzelnen Dezimalstellen
EDBD-   LDY   #$00       als Zeiger auf Integer-Konstante  -100 000 000
EDBF-   LDX   #$80       als Zähler (Bit7=1 bedeutet negative Konstante)
EDC1-   LDA   $A1        Konstante zu FAC1 addieren
```

```
EDC3-   CLC
EDC4-   ADC   $EE6C,Y
EDC7-   STA   $A1        M4
EDC9-   LDA   $A0
EDCB-   ADC   $EE6B,Y
EDCE-   STA   $A0        M3
EDD0-   LDA   $9F
EDD2-   ADC   $EE6A,Y
EDD5-   STA   $9F        M2
EDD7-   LDA   $9E
EDD9-   ADC   $EE69,Y
EDDC-   STA   $9E        M1
EDDE-   INX              Zähler erhöhen
EDDF-   BCS   $EDE5      weiter mit dieser Konstanten, falls Ergebnis positiv
EDE1-   BPL   $EDC1      und Konstante negativ oder umgekehrt
EDE3-   BMI   $EDE7
EDE5-   BMI   $EDC1
EDE7-   TXA              Zähler = Ziffer +1
EDE8-   BCC   $EDEE      d.h. Konstante war negativ
EDEA-   EOR   #$FF       sonst bzgl. 10 komplementieren, d.h.  Accu = 10 - Accu
EDEC-   ADC   #$0A
EDEE-   ADC   #$2F       ergibt Ascii-Kode der Ziffer
EDF0-   INY              Zeiger auf nächste Konstante erhöhen
EDF1-   INY
EDF2-   INY
EDF3-   INY
EDF4-   STY   $83        und merken
EDF6-   LDY   $AD        Zeiger auf den String holen
EDF8-   INY
EDF9-   TAX              Ziffernkode merken
EDFA-   AND   #$7F       Bit7 löschen
EDFC-   STA   $00FF,Y    und in String eintragen
EDFF-   DEC   $99        Anzahl weiterer Vorkommastellen verringern
EE01-   BNE   $EE09      Dezimalpunkt noch nicht erreicht
EE03-   LDA   #$2E       sonst Ascii-Kode für Dezimalpunkt
EE05-   INY
EE06-   STA   $00FF,Y    in String eintragen
EE09-   STY   $AD        Zeiger auf den String wieder retten
EE0B-   LDY   $83        Zeiger auf Konstanten-Tabelle holen
EE0D-   TXA              Ziffer, Bit7 = Vorzeichen der alten Konstante
EE0E-   EOR   #$FF       invertieren
EE10-   AND   #$80       neue Konstante hat umgekehrtes Vorzeichen
EE12-   TAX              X wieder als Zähler
EE13-   CPY   #$24       sämtliche 9 Stellen bestimmt?
EE15-   BNE   $EDC1      nein, neue Ziffer bestimmen

unnötige Nullen entfernen
EE17-   LDY   $AD        Zeiger auf den String holen
EE19-   LDA   $00FF,Y    suche hinterstes von "0" verschiedenes Zeichen
EE1C-   DEY
EE1D-   CMP   #$30       Ascii-kode von "0"
EE1F-   BEQ   $EE19      "0", weiter
EE21-   CMP   #$2E       Dezimalpunkt?
EE23-   BEQ   $EE26      ja, keine Nachkommastelle, Zeiger auf Dezimalpunkt
EE25-   INY              sonst Zeiger auf erste "0" setzen

Format für Exponentialdarstellung herstellen
EE26-   LDA   #$2B       Ascii-Kode für "+"
EE28-   LDX   $9A        Exponent
EE2A-   BEQ   $EE5A      keine Exponential-Darstellung
EE2C-   BPL   $EE36      Exponent positiv
```

```
EE2E-   LDA   #$00
EE30-   SEC
EE31-   SBC   $9A          ergibt Betrag des Exponenten
EE33-   TAX
EE34-   LDA   #$2D         Ascii-Kode für "-"
EE36-   STA   $0101,Y      in String eintragen (mit Lücke für "E")
EE39-   LDA   #$45         Ascii-Kode für "E"
EE3B-   STA   $0100,Y      in String eintragen
EE3E-   TXA                Betrag des Exponenten
EE3F-   LDX   #$2F         X als Zähler vorbereiten
EE41-   SEC
EE42-   INX
EE43-   SBC   #$0A         so oft 10 subtrahieren, bis Ergebnis negativ,
EE45-   BCS   $EE42        X enthält dann den Ascii-Kode der Zehnerstelle
EE47-   ADC   #$3A         (Einerstelle -10) + 58 ergibt Ascii-Kode Einerstelle
EE49-   STA   $0103,Y      Einerstelle des Exponenten eintragen
EE4C-   TXA
EE4D-   STA   $0102,Y      Zehnerstelle des Exponenten eintragen
EE50-   LDA   #$00         als String-Endmarke
EE52-   STA   $0104,Y      eintragen
EE55-   BEQ   $EE5F        Stringzeiger holen, fertig

Einsprung falls Zahl =0 war
EE57-   STA   $00FF,Y      Ascii-Kode für "0" eintragen
EE5A-   LDA   #$00         String-Endmarke anfügen
EE5C-   STA   $0100,Y
EE5F-   LDA   #$00         Zeiger auf Zahlenstring
EE61-   LDY   #$01
EE63-   RTS
--------------------

EE64-   80 00 00 00 00     FP-Konstante  0.5
--------------------

Konstanten im Integerformat
EE69-   FA 0A 1F 00          - 100 000 000
EE6D-   00 98 96 80             10 000 000
EE71-   FF F0 BD C0           - 1 000 000
EE75-   00 01 86 A0               100 000
EE79-   FF FF D8 F0             - 10 000
EE7D-   00 00 03 E8                1 000
EE81-   FF FF FF 9C              - 100
EE85-   00 00 00 0A                 10
EE89-   FF FF FF FF                 - 1
--------------------

Applesoft-Funktion  SQR
EE8D-   JSR   $EB63        FAC1 nach FAC2 übertragen
EE90-   LDA   #$64         Zeiger auf FP-Konstante 0.5
EE92-   LDY   #$EE
EE94-   JSR   $EAF9        Konstante nach FAC1 übertragen, weiter mit Potenzierung

FAC1 = FAC2 ^ FAC1
mit der Formel  a ^ b = exp( b * ln(a) )
I: Accu enthält Exponenten von FAC1
EE97-   BEQ   $EF09        FAC1 = 0, weiter bei EXP ergibt richtiges Ergebnis 1
EE99-   LDA   $A5          Exponent von FAC2
EE9B-   BNE   $EEA0        ungleich 0
EE9D-   JMP   $E850        sonst Ergebnis = 0, --> FAC1 nullsetzen
EEA0-   LDX   #$8A         Zeiger auf FAC5
EEA2-   LDY   #$00
```

```
EEA4-   JSR   $EB2B     FAC1 nach FAC2 übertragen  (=b)
EEA7-   LDA   $AA       Vorzeichen von FAC2  (=a)
EEA9-   BPL   $EEBA     positiv
EEAB-   JSR   $EC23     FAC1 = INT (FAC1)  (ergibt Int(b))
EEAE-   LDA   #$8A      Zeiger auf FAC5
EEB0-   LDY   #$00
EEB2-   JSR   $EBB2     FAC1 = FAC5? (gleich falls b ganzzahlig, -> A=0, Y=4)
EEB5-   BNE   $EEBA     ungleich, d.h. b nicht ganzzahlig (A=255, Y=1...4)
EEB7-   TYA             damit Bit7=0  (für positives Vorzeichen von a)
EEB8-   LDY   $0D       =b, ganzzahlig   (vgl $EC23)
EEBA-   JSR   $EB55     FAC2 --> FAC1, negativ falls a<0 und b nicht ganzzahlig
EEBD-   TYA             =0 falls a>0;  =b falls a<0 und b ganz;  =1..4 sonst
EEBE-   PHA             retten
EEBF-   JSR   $E941     LOG-Routine --> ln(a) bzw. ln(-a) falls a<0 und b ganz
EEC2-   LDA   #$8A                                  / sonst ILLEGAL QUANTITY
EEC4-   LDY   #$00      Zeiger auf FAC5  (enthält b)
EEC6-   JSR   $E97F     FAC1 = FAC1 * FAC5 , ergibt  b * ln(a)
EEC9-   JSR   $EF09     FAC1 = EXP(FAC1), ergibt  exp( b * ln(a) )
EECC-   PLA             Bit0=1 falls  a<0 und b ungeradzahlig  (vgl. $EEBD)
EECD-   LSR             Bit0 --> C
EECE-   BCC   $EEDA     Vorzeichen des Ergebnisses positiv, fertig
EED0-   LDA   $9D       Exponent von FAC1
EED2-   BEQ   $EEDA     Ergebnis =0, fertig
EED4-   LDA   $A2       negatives Vorzeichen eintragen
EED6-   EOR   #$FF
EED8-   STA   $A2
EEDA-   RTS
---------------------

EEDB-   81 38 AA 3B 29     FP-Konstante  1 / ln(2)
---------------------

Reihenkoeffizienten für  EXP-Routine
EEE0-   07                 Grad des Polynoms
EEE1-   71 34 58 3E 56     2.14987637 E-5
EEE6-   74 16 7E B3 1B     1.43523140 E-4
EEEB-   77 2F EE E3 85     1.34226348 E-3
EEF0-   7A 1D 84 1C 2A     9.61401701 E-3
EEF5-   7C 63 59 58 0A     5.55051269 E-2
EEFA-   7E 75 FD E7 C6     2.40226385 E-1
EEFF-   80 31 72 18 10     6.93147186 E-1 = ln(2)
EF04-   81 00 00 00 00     1.0
---------------------

Applesoft-Funktion  EXP
mit der Formel  exp(x) = 2 ^ ( x / ln(2) )
EF09-   LDA   #$DB      Zeiger auf FP-Konstante  1 / ln(2)
EF0B-   LDY   #$EE
EF0D-   JSR   $E97F     FAC1 = Konstante * FAC1, ergibt also  x / ln(2)
EF10-   LDA   $AC       Rundungsstelle von FAC1
EF12-   ADC   #$50
EF14-   BCC   $EF19     abrunden
EF16-   JSR   $EB7A     aufrunden
EF19-   STA   $92       neue Rundungsstelle merken
EF1B-   JSR   $EB66     FAC1 nach FAC2 kopieren
EF1E-   LDA   $9D       Exponent von FAC1
EF20-   CMP   #$88      vergleichen
EF22-   BCC   $EF27     FAC1 < 128
EF24-   JSR   $EA2B     Argument zu groß, --> OVERFLOW ERROR
EF27-   JSR   $EC23     INT-Routine
EF2A-   LDA   $0D       ganzzahliger Anteil von FAC1
```

```
EF2C-   CLC
EF2D-   ADC   #$81
EF2F-   BEQ   $EF24       FAC1 >= 127, OVERFLOW ERROR
EF31-   SEC
EF32-   SBC   #$01        minus 1
EF34-   PHA               ergibt  Int( x / ln(2) ) - 1, retten
EF35-   LDX   #$05        FAC1 mit FAC2 vertauschen
EF37-   LDA   $A5,X
EF39-   LDY   $9D,X
EF3B-   STA   $9D,X
EF3D-   STY   $A5,X
EF3F-   DEX
EF40-   BPL   $EF37       nächstes Byte vertauschen
EF42-   LDA   $92         gemerkte Rundungsstelle
EF44-   STA   $AC         in FAC1 eintragen
EF46-   JSR   $E7AA       FAC1 = FAC2 - FAC1, ergibt negativen Bruchteil
EF49-   JSR   $EED0       FAC1 = -FAC1, ergibt dezimalen Bruchteil von x / ln(2)
EF4C-   LDA   #$E0        Zeiger auf Tabelle der Polynomkoeffizienten
EF4E-   LDY   #$EE        (diese betragen etwa  ln(2)^k / k!, k=0,1,...,7)
EF50-   JSR   $EF72       Polynom auswerten mit Argument x', ergibt  exp(x')
EF53-   LDA   #$00
EF55-   STA   $AB         Kombivorzeichen löschen, d.h. "+"
EF57-   PLA               Exponent bzgl.2, = Int( x / ln(2) ) -1
EF58-   JSR   $EA10       zu Exponent von FAC1 addiert ergibt gesamten Exponenten
EF5B-   RTS
--------------------

Polynom auswerten (nur ungerade Potenzen)
I: FAC1 enthält das Argument x
EF5C-   STA   $AD         Zeiger auf Tabelle der Polynomkoeffizienten
EF5E-   STY   $AE
EF60-   JSR   $EB21       FAC1 gerundet nach FAC3 übertragen
EF63-   LDA   #$93        Zeiger auf FAC3
EF65-   JSR   $E97F       FAC1 = FAC1 * FAC3, ergibt also  x^2
EF68-   JSR   $EF76       Polynom auswerten, ergibt  K1 + K3*x^2 + K5*x^4 +...
EF6B-   LDA   #$93        Zeiger auf FAC3
EF6D-   LDY   #$00        FAC1 = FAC1 * FAC3 ergibt  K1*x + K3*x^3 + K5*x^5 +...
EF6F-   JMP   $E97F       FAC1 = FAC1 * FAC3, ergibt  K1*x + K3*x^3 + K5*x^5 +...
--------------------

Polynom nach Hornerschema auswerten
I: FAC1 enthält das Argument x
   A,Y zeigt auf Tabelle mit den Polynomkoeffizienten
O: FAC1 enthält das Ergebnis
EF72-   STA   $AD         Zeiger auf Tabelle der Polynomkoeffizienten
EF74-   STY   $AE
EF76-   JSR   $EB1E       FAC1 nach FAC4 übertragen
EF79-   LDA   ($AD),Y     Polynomgrad holen (als Zähler)
EF7B-   STA   $A3         eintragen
EF7D-   LDY   $AD
EF7F-   INY               Zeiger auf 1. Koeffizienten, LByte
EF80-   TYA               --> A
EF81-   BNE   $EF85
EF83-   INC   $AE         HByte korrigieren
EF85-   STA   $AD         LByte eintragen
EF87-   LDY   $AE         damit zeigt A,Y auf 1. Koeffizienten
EF89-   JSR   $E97F       FAC1 = FAC1 * (A,Y), d.h. mit x multiplizieren
EF8C-   LDA   $AD         Zeiger auf Koeffizienten
EF8E-   LDY   $AE
EF90-   CLC
EF91-   ADC   #$05        um 5 erhöhen
```

```
EF93-   BCC   $EF96
EF95-   INY
EF96-   STA   $AD        ergibt Zeiger auf nächsten Koeffizienten
EF98-   STY   $AE
EF9A-   JSR   $E7BE      FAC1 = FAC1 + (A,Y), d.h. neuen Koeff. addieren
EF9D-   LDA   #$98       Zeiger auf FAC4 (enthält x)
EF9F-   LDY   #$00
EFA1-   DEC   $A3        Zähler dekrementieren
EFA3-   BNE   $EF89      Absolutglied noch nicht erreicht, weiter in Schleife
EFA5-   RTS
--------------------

Konstanten für  RND
EFA6-   98 35 44 7A (68)     1.18795464 E+7
EFAA-   68 28 B1 46 (20)     3.92767778 E-8
--------------------

Applesoft-Funktion  RND
EFAE-   JSR   $EB82      Vorzeichen von FAC1 testen
EFB1-   TAX              = 255 / 0 / 1   bei  - / 0 / +
EFB2-   BMI   $EFCC      negativ
EFB4-   LDA   #$C9       Zeiger auf frühere Zufallszahl
EFB6-   LDY   #$00
EFB8-   JSR   $EAF9      diese nach FAC1 übertragen
EFBB-   TXA
EFBC-   BEQ   $EFA5      Argument war 0, fertig  (keine neue Zufallszahl)
EFBE-   LDA   #$A6       Zeiger auf 1. Konstante (Z1)
EFC0-   LDY   #$EF
EFC2-   JSR   $E97F      FAC1 = FAC1 * Z1
EFC5-   LDA   #$AA       Zeiger auf 2. Konstante (Z2)
EFC7-   LDY   #$EF
EFC9-   JSR   $E7BE      FAC1 = FAC1 + Z2

FAC1 durcheinander schütteln
EFCC-   LDX   $A1        M4 mit M1 vertauschen
EFCE-   LDA   $9E
EFD0-   STA   $A1
EFD2-   STX   $9E
EFD4-   LDA   #$00
EFD6-   STA   $A2        erzwingt positives Vorzeichen
EFD8-   LDA   $9D        Exponent
EFDA-   STA   $AC        als Rundungsstelle verwenden
EFDC-   LDA   #$80       als neuer Exponent
EFDE-   STA   $9D        eintragen, Ergebnis damit sicher < 1
EFE0-   JSR   $E82E      FAC1 linksbündig machen
EFE3-   LDX   #$C9       Zeiger auf Register für Zufallszahl
EFE5-   LDY   #$00
EFE7-   JMP   $EB2B      neue Zufallszahl dort eintragen
--------------------

Applesoft-Funktion  COS
EFEA-   LDA   #$66       Zeiger auf FP-Konstante  Pi / 2
EFEC-   LDY   #$F0
EFEE-   JSR   $E7BE      FAC1 = FAC1 + Pi / 2, weiter bei SIN-Routine

Applesoft-Funktion  SIN
EFF1-   JSR   $EB63      FAC1 gerundet nach FAC2
EFF4-   LDA   #$6B       Zeiger auf FP-Konstante  2 * Pi
EFF6-   LDY   #$F0
EFF8-   LDX   $AA        Vorzeichen von FAC2 holen
EFFA-   JSR   $EA5E      FAC1 = FAC2 / (2 * Pi), d.h. Periode normieren
```

```
EFFD-   JSR   $EB63     FAC1 gerundet nach FAC2
F000-   JSR   $EC23     INT-Routine bestimmt Anzahl voller Perioden
F003-   LDA   #$00
F005-   STA   $AB       Kombi-Vorzeichen löschen
F007-   JSR   $E7AA     FAC1 = FAC2 - FAC1   (damit innerhalb 1. Periode)

Argument in Intervall (-Pi/2, Pi/2) transformieren
F00A-   LDA   #$70      Zeiger auf FP-Konstante  0.25
F00C-   LDY   #$F0
F00E-   JSR   $E7A7     FAC1 = 0.25 - FAC1
F011-   LDA   $A2       Vorzeichen von FAC1
F013-   PHA             merken
F014-   BPL   $F023     positiv, Argument war im 1. Quadranten
F016-   JSR   $E7A0     FAC1 = FAC1 + 0.5
F019-   LDA   $A2       Vorzeichen holen
F01B-   BMI   $F026     negativ, Argument war im 4. Quadranten
F01D-   LDA   $16       Vorzeichen-Flag für TAN-Routine
F01F-   EOR   #$FF      negativ, da cos im 2. und 3. Quadranten negativ ist
F021-   STA   $16
F023-   JSR   $EED0     FAC1 = -FAC1
F026-   LDA   #$70      Zeiger auf Konstante  0.25
F028-   LDY   #$F0
F02A-   JSR   $E7BE     FAC1 = FAC1 + 0.25
F02D-   PLA             Bit7=0 falls Argument im 1. Quadranten
F02E-   BPL   $F033     Vorzeichen nicht wechseln
F030-   JSR   $EED0     FAC1 = -FAC1
F033-   LDA   #$75      Zeiger auf Tabelle mit Polynomkoeffizienten
F035-   LDY   #$F0      (diese betragen ungefähr +-(2*Pi)^k / k!, k=1,3,...,11)
F037-   JMP   $EF5C     Polynom auswerten, fertig
--------------------

Applesoft-Funktion  TAN
F03A-   JSR   $EB21     FAC1 nach FAC3 übertragen
F03D-   LDA   #$00
F03F-   STA   $16       Vorzeichen-Flag auf "+" setzen
F041-   JSR   $EFF1     FAC3 = FAC1; FAC1 = SIN(FAC1)
F044-   LDX   #$8A      Zeiger auf FAC5
F046-   LDY   #$00
F048-   JSR   $EFE7     FAC1 gerundet nach FAC5 übertragen
F04B-   LDA   #$93      Zeiger auf FAC3 (normiertes Argument)
F04D-   LDY   #$00
F04F-   JSR   $EAF9     FAC3 nach FAC1 übertragen
F052-   LDA   #$00
F054-   STA   $A2       Vorzeichen von FAC1 "+" setzen
F056-   LDA   $16       Vorzeichen-Flag (enthält Vorzeichen für Cos-Funktion)
F058-   JSR   $F062     FAC1 = cos(FAC1)
F05B-   LDA   #$8A      Zeiger auf FAC5
F05D-   LDY   #$00
F05F-   JMP   $EA66     FAC1 = FAC5 / FAC1 (d.h. sinx / cosx = tanx)
--------------------
F062-   PHA             Vorzeichen-Flag retten
F063-   JMP   $F023     FAC1 = sin(0.25 - FAC1) mit normierten Argumenten
--------------------
F066-   81 49 0F DA A2    FP-Konstante  Pi / 2
F06B-   83 49 0F DA A2    FP-Konstante  2 * Pi
F070-   7F 00 00 00 00    FP-Konstante  0.25
--------------------

Reihenkoeffizienten für  SIN / COS / TAN
F075-   05                Polynomgrad
F076-   84 E6 1A 2D 1B    -14.3813907
```

```
F07B-   86 28 07 FB F8      42.0077971
F080-   87 99 68 89 01     -76.7041703
F085-   87 23 35 DF E1      81.6052237
F08A-   86 A5 5D E7 28     -41.3417021
F08F-   83 49 0F DA A2      2 * Pi
---------------------

Ascii für "MICROSOFT!", jedoch Bits 0,1,2 invertiert
F094-   A6 D3 C1 C8 D4 C8 D5 C4 CE CA
---------------------

Applesoft-Funktion  ATN
F09E-   LDA   $A2        Vorzeichen von FAC1 retten
F0A0-   PHA
F0A1-   BPL   $F0A6      positiv
F0A3-   JSR   $EED0      sonst FAC1 = -FAC1,  d.h. Betrag vom Argument
F0A6-   LDA   $9D        Exponent von FAC1
F0A8-   PHA              retten
F0A9-   CMP   #$81       vergleichen
F0AB-   BCC   $F0B4      FAC1 < 1
F0AD-   LDA   #$13       sonst Kehrwert bilden: Zeiger auf Konstante 1 holen
F0AF-   LDY   #$E9
F0B1-   JSR   $EA66      FAC1 = 1 / FAC1
F0B4-   LDA   #$CE       Zeiger auf Tabelle mit Reihenkoeffizienten
F0B6-   LDY   #$F0
F0B8-   JSR   $EF5C      Polynom auswerten
F0BB-   PLA              gemerkter Exponent
F0BC-   CMP   #$81       wieder vergleichen
F0BE-   BCC   $F0C7      Argument war < 1
F0C0-   LDA   #$66       sonst Ergebnis korrigieren:
F0C2-   LDY   #$F0       Zeiger auf Konstante  Pi / 2
F0C4-   JSR   $E7A7      FAC1 = Pi/2 - FAC1
F0C7-   PLA              gemerktes Vorzeichen
F0C8-   BPL   $F0CD      positiv, fertig
F0CA-   JMP   $EED0      FAC1 = -FAC1
F0CD-   RTS
---------------------

Reihenkoeffizienten für  ATN
F0CE-   0B                 Polynomgrad = 11
F0CF-   76 B3 83 BD D3     -6.84793912 E-4
F0D4-   79 1E F4 A6 F5      4.85094216 E-3
F0D9-   7B 83 FC B0 10     -1.61117018 E-2
F0DE-   7C 0C 1F 67 CA      3.42096380 E-2
F0E3-   7C DE 53 CB C1     -5.42791328 E-2
F0E8-   7D 14 64 70 4C      7.24571965 E-2
F0ED-   7D B7 EA 51 7A     -8.98023954 E-2
F0F2-   7D 63 30 88 7E      1.10932413 E-1
F0F7-   7E 92 44 99 3A     -1.42839808 E-1
F0FC-   7E 4C CC 91 C7      1.99999120 E-1
F101-   7F AA AA AA 13     -3.33333157 E-1
F106-   81 00 00 00 00      1.0
---------------------

Kopie der CHRGET-Routine
O: C=0 falls Ziffer;  Z=1 falls Kode = 00 oder 3A
F10B-   INC   $B8        CHRGET-Zeiger erhöhen
F10D-   BNE   $F111
F10F-   INC   $B9
F111-   LDA   $EA60      Dummy Adresse, später steht dort der CHRGET-Zeiger
F114-   CMP   #$3A       Ascii für ":"
```

```
F116-   BCS   $F122      Zeichen hinter den Ziffern  (also keine Ziffer)
F118-   CMP   #$20       Ascii für Leerzeichen
F11A-   BEQ   $F10B      Leerzeichen ignorieren, nächstes Zeichen holen
F11C-   SEC
F11D-   SBC   #$30
F11F-   SEC
F120-   SBC   #$D0       damit C=1 falls Kode < 30  (also falls keine Ziffer)
F122-   RTS
--------------------
F123-   80 4F C7 52 58     FP-Konstante 0.811635157
--------------------

Kaltstart-Routine
F128-   LDX   #$FF
F12A-   STX   $76        Direkt-Modus setzen
F12C-   LDX   #$FB
F12E-   TXS              Stack initialisieren
F12F-   LDA   #$28       Adresse der Kaltstart-Routine
F131-   LDY   #$F1
F133-   STA   $01        nach $0001
F135-   STY   $02
F137-   STA   $04        und $0004
F139-   STY   $05
F13B-   JSR   $F273      NORMAL durchführen (normale Bildschirmausgabe)
F13E-   LDA   #$4C       Assembler-Kode für JMP absolut
F140-   STA   $00        vor die Sprungadressen in der Zeropage setzen
F142-   STA   $03
F144-   STA   $90
F146-   STA   $0A
F148-   LDA   #$99       Zeiger auf ILLEGAL QUANTITY ERROR -Ausgabe
F14A-   LDY   #$E1
F14C-   STA   $0B        als USR-Adresse eintragen
F14E-   STY   $0C
F150-   LDX   #$1C       als Zähler
F152-   LDA   $F10A,X    CHRGET-Routine und 'Zufallszahl' in Zeropage kopieren
F155-   STA   $B0,X
F157-   STX   $F1        bei letztem Durchgang X=1 als SPEED-Flag eintragen
F159-   DEX
F15A-   BNE   $F152      weiter in Schleife
F15C-   STX   $F2        TRACE-Flag löschen
F15E-   TXA
F15F-   STA   $A4        Hilfsregister löschen
F161-   STA   $54        Adresse des obersten Deskriptors auf Deskr.-Stack
F163-   PHA              0 auf Stack
F164-   LDA   #$03
F166-   STA   $8F        Deskriptor-Länge für Garbage Collection vorgeben
F168-   JSR   $DAFB      CR ausgeben
F16B-   LDA   #$01
F16D-   STA   $01FD      1 in Stack eintragen (oberhalb der 0 von $F163)
F170-   STA   $01FC
F173-   LDX   #$55       als Zeiger für Deskriptoren-Stack auf dessen Anfang
F175-   STX   $52        eintragen
F177-   LDA   #$00       Zeiger für RAM-Test auf $0800 setzen
F179-   LDY   #$08
F17B-   STA   $50
F17D-   STY   $51

höchste verfügbare RAM-Adresse suchen
F17F-   LDY   #$00
F181-   INC   $51        Zeiger erhöhen
F183-   LDA   ($50),Y    $0900, $0A00, etc. holen
```

```
F185-   EOR   #$FF        invertieren
F187-   STA   ($50),Y     und in Speicherzelle eintragen
F189-   CMP   ($50),Y     prüfen, ob Inhalt verändert wurde
F18B-   BNE   $F195       nein, RAM-Ende gefunden
F18D-   EOR   #$FF        nochmals invertieren
F18F-   STA   ($50),Y     erneut eintragen
F191-   CMP   ($50),Y     und vergleichen
F193-   BEQ   $F181       Inhalt wurde wieder verändert, noch nicht RAM-Ende
F195-   LDY   $50         Zeiger auf gefundenes RAM-Ende
F197-   LDA   $51
F199-   AND   #$F0        auf Ende des letzten 4K-Blocks setzen
F19B-   STY   $73         als HIMEM, d.h. höchste benutzte RAM-Adresse eintragen
F19D-   STA   $74
F19F-   STY   $6F         und als Anfang des Stringbereichs (keine Strings da)
F1A1-   STA   $70
F1A3-   LDX   #$00        Zeiger auf $0800
F1A5-   LDY   #$08
F1A7-   STX   $67         als vorläufiger Anfang für Programmtext
F1A9-   STY   $68
F1AB-   LDY   #$00
F1AD-   STY   $D6         Autostart-Flag löschen
F1AF-   TYA
F1B0-   STA   ($67),Y     $0800 mit 0 besetzen
F1B2-   INC   $67         Programm-Anfangszeiger auf $0801 erhöhen
F1B4-   BNE   $F1B8
F1B6-   INC   $68
F1B8-   LDA   $67         Programm-Anfangszeiger
F1BA-   LDY   $68
F1BC-   JSR   $D3E3       unterhalb HIMEM?  Falls nein --> OUT OF MEMORY ERROR
F1BF-   JSR   $D64B       NEW, CLEAR, Stackinitialisierung durchführen
F1C2-   LDA   #$3A        Zeiger auf String-Druckroutine
F1C4-   LDY   #$DB
F1C6-   STA   $04         in JMP-Befehl ab $0003 eintragen
F1C8-   STY   $05
F1CA-   LDA   #$3C        Zeiger auf Warmstart
F1CC-   LDY   #$D4
F1CE-   STA   $01         in JMP-Befehl ab $0000 eintragen
F1D0-   STY   $02
F1D2-   JMP   ($0001)     indirekter Sprung zu Warmstart
---------------------

Applesoft-Routine  CALL
F1D5-   JSR   $DD67       numerischen Term auswerten  (Sprungadresse)
F1D8-   JSR   $E752       FAC1 in Integer (in 50,51) umwandeln
F1DB-   JMP   ($0050)     Sprung zur Assembler-Routine
---------------------

Applesoft-Routine  IN#
F1DE-   JSR   $E6F8       1-Byte-Integer aus Programmtext holen --> X
F1E1-   TXA
F1E2-   JMP   $FE8B       Monitor-Routine IN#  definiert Eingabekanal
---------------------

Applesoft-Routine  PR#
F1E5-   JSR   $E6F8       1-Byte-Integer aus Programmtext holen --> X
F1E8-   TXA
F1E9-   JMP   $FE95       Monitor-Routine PR#  definiert Ausgabekanal
---------------------

Plot-Parameter für LoRes-Grafik holen
O: 1. Parameter in F0;  2. Parameter in 2C = 2D = X
```

```
F1EC-   JSR   $E6F8      1-Byte-Integer aus Programmtext holen --> X
F1EF-   CPX   #$30       mit 48 vergleichen
F1F1-   BCS   $F206      auf jeden Fall zu groß, --> ILLEGAL QUANTITY ERROR
F1F3-   STX   $F0        sonst 1. Parameter eintragen
F1F5-   LDA   #$2C       Ascii für Komma
F1F7-   JSR   $DEC0      muß folgen, sonst SYNTAX ERROR
F1FA-   JSR   $E6F8      1-Byte-Integer aus Programmtext holen --> X
F1FD-   CPX   #$30       vergleichen mit 48
F1FF-   BCS   $F206      auf jeden Fall zu groß, --> ILLEGAL QUANTITY ERROR
F201-   STX   $2C        sonst 2. Parameter eintragen
F203-   STX   $2D
F205-   RTS
--------------------
F206-   JMP   $E199      ILLEGAL QUANTITY ERROR ausgeben
--------------------

hole Parameter für  HLIN / VLIN
O: Parameter in F0, 2C = 2D, X; dabei F0 < 2C/2D
F209-   JSR   $F1EC      hole 2 Parameter aus Programmtext --> F0,2C=2D=X
F20C-   CPX   $F0        Parameter vergleichen
F20E-   BCS   $F218      Parameter in F0 kleiner, ok
F210-   LDA   $F0        sonst vertauschen
F212-   STA   $2C
F214-   STA   $2D
F216-   STX   $F0
F218-   LDA   #$C5       Token für AT
F21A-   JSR   $DEC0      muß folgen, sonst SYNTAX ERROR
F21D-   JSR   $E6F8      3. Parameter aus Programmtext holen
F220-   CPX   #$30       mit 48 vergleichen
F222-   BCS   $F206      zu groß, --> ILLEGAL QUANTITY ERROR
F224-   RTS
--------------------

Applesoft-Routine  PLOT
F225-   JSR   $F1EC      PLOT-Parameter aus Programmtext holen
F228-   TXA              2. Parameter = Y-Koordinate (Wert schon geprüft)
F229-   LDY   $F0        1. Parameter = X-Koordinate (muß < 40 sein)
F22B-   CPY   #$28       vergleichen
F22D-   BCS   $F206      zu groß, --> ILLEGAL QUANTITY ERROR
F22F-   JMP   $F800      sonst zur Monitor-Routine  PLOT
--------------------

Applesoft-Routine  HLIN
F232-   JSR   $F209      hole 3 Parameter aus Programmtext
F235-   TXA              3. Parameter = Y-Koordinate
F236-   LDY   $2C        X-Koordinate des rechten Endes der Linie
F238-   CPY   #$28       muß kleiner 40 sein
F23A-   BCS   $F206      zu groß, --> ILLEGAL QUANTITY ERROR
F23C-   LDY   $F0        X-Koordinate des linken Endes (sicher < rechtes Ende)
F23E-   JMP   $F819      zur Monitor-Routine  HLINE
--------------------

Applesoft-Routine  VLIN
F241-   JSR   $F209      hole 3 Parameter aus Programmtext
F244-   TXA              3. Parameter = X-Koordinate
F245-   TAY              nach Y, muß kleiner als 40 sein
F246-   CPY   #$28       vergleichen
F248-   BCS   $F206      zu groß, --> ILLEGAL QUANTITY ERROR
F24A-   LDA   $F0        Y-Koord. des oberen Linienendes (2C=2D unteres Ende)
F24C-   JMP   $F828      zur Monitor-Routine  VLINE
--------------------
```

```
Applesoft-Routine  COLOR
F24F-   JSR   $E6F8      1-Byte-Integer aus Programmtext holen --> X
F252-   TXA              COLOR-Wert
F253-   JMP   $F864      zur Monitor-Routine  SETCOL
--------------------

Applesoft-Routine  VTAB
F256-   JSR   $E6F8      1-Byte-Integer  aus Programmtext holen --> X
F259-   DEX              minus 1, da oberste Zeile den Wert 0 haben soll
F25A-   TXA              Bildschirmzeile, muß kleiner 24 sein
F25B-   CMP   #$18       vergleichen
F25D-   BCS   $F206      zu groß, --> ILLEGAL QUANTITY ERROR
F25F-   JMP   $FB5B      zur Monitor-Routine  TABV
--------------------

Applesoft-Routine  SPEED
F262-   JSR   $E6F8      1-Byte-Integer aus Programmtext holen --> X
F265-   TXA              0 = langsam, 255 = schnell
F266-   EOR   #$FF       invertieren
F268-   TAX
F269-   INX              plus 1 , ergibt WAIT-Parameter
F26A-   STX   $F1        ins SPEED-Flag eintragen  (1 = maximal schnell)
F26C-   RTS
--------------------

Applesoft-Routinen  TRACE / NOTRACE
F26D-   SEC              Einsprung für  TRACE
F26E-   BCC   $F288      --> F26F- CLC , Einsprung für  NOTRACE
F270-   ROR   $F2        C --> Bit7 vom  TRACE-Flag
F272-   RTS
--------------------

Applesoft-Routinen  NORMAL / INVERSE
F273-   LDA   #$FF       Einsprung für  NORMAL, hole NORMAL-Maske
F275-   BNE   $F279      immer springen
F277-   LDA   #$3F       Einsprung für  INVERSE, hole INVERSE-Maske
F279-   LDX   #$00       um das FLASH-Flag zu löschen
F27B-   STA   $32        INVERSE-Flag eintragen
F27D-   STX   $F3        FLASH-Flag eintragen
F27F-   RTS
--------------------

Applesoft-Routine  FLASH
F280-   LDA   #$7F       FLASH-Maske holen, kommt ins INVERSE-Flag
F282-   LDX   #$40       um das FLASH-Flag zu setzen
F284-   BNE   $F27B      Flags eintragen
--------------------

Applesoft-Routine  HIMEM:
F286-   JSR   $DD67      numerischen Ausdruck auswerten --> FAC1
F289-   JSR   $E752      FAC1 in Integer (in 50,51) umwandeln
F28C-   LDA   $50        50,51 mit Feldvariablen-Ende +1  vergleichen
F28E-   CMP   $6D
F290-   LDA   $51
F292-   SBC   $6E
F294-   BCS   $F299      liegt nicht darunter, ok.
F296-   JMP   $D410      sonst SYNTAX ERROR
F299-   LDA   $50
F29B-   STA   $73        HIMEM:-Adresse eintragen
F29D-   STA   $6F        als Anfang des Stringbereichs eintragen
```

```
F29F-   LDA   $51        HBytes eintragen
F2A1-   STA   $74
F2A3-   STA   $70
F2A5-   RTS
--------------------

Applesoft-Routine  LOMEM:
F2A6-   JSR   $DD67      numerischen Ausdruck auswerten  --> FAC1
F2A9-   JSR   $E752      FAC1 in Integer (in 50,51) umwandeln
F2AC-   LDA   $50        50,51 mit HIMEM-Wert vergleichen
F2AE-   CMP   $73
F2B0-   LDA   $51
F2B2-   SBC   $74
F2B4-   BCS   $F296      liegt nicht darunter, --> SYNTAX ERROR
F2B6-   LDA   $50        50,51 mit bisherigem LOMEM-Wert vergleichen
F2B8-   CMP   $69
F2BA-   LDA   $51
F2BC-   SBC   $6A
F2BE-   BCC   $F296      liegt darunter, --> SYNTAX ERROR
F2C0-   LDA   $50        neuen LOMEM-Wert eintragen
F2C2-   STA   $69
F2C4-   LDA   $51
F2C6-   STA   $6A
F2C8-   JMP   $D66C      CLEAR und Stackinitialisierung durchführen
--------------------

Applesoft-Routine  ONERR
F2CB-   LDA   #$AB       Token für GOTO
F2CD-   JSR   $DEC0      muß folgen
F2D0-   LDA   $B8        CHRGET-Zeiger
F2D2-   STA   $F4        als Adresse des gültigen ONERR-Statements
F2D4-   LDA   $B9
F2D6-   STA   $F5
F2D8-   SEC
F2D9-   ROR   $D8        C=1 --> Bit7 des ONERR-Flags, d.h. ONERR aktivieren
F2DB-   LDA   $75        momentane Zeilennummer
F2DD-   STA   $F6        als Zeilennummer des gültigen ONERR-Statements
F2DF-   LDA   $76
F2E1-   STA   $F7
F2E3-   JSR   $D9A6      suche nächste Programmzeile
F2E6-   JMP   $D998      CHRGET-Zeiger setzen, weiter im Programm
--------------------

ONERR GOTO- Fehlerauswertung
I: X enthält Fehlerart-Kode
F2E9-   STX   $DE        Fehlerart-Kode merken
F2EB-   LDX   $F8        Zwischenspeicher für Stackpointer
F2ED-   STX   $DF        Stackpointer retten
F2EF-   LDA   $75        momentane Zeilennummer für RESUME retten
F2F1-   STA   $DA
F2F3-   LDA   $76
F2F5-   STA   $DB
F2F7-   LDA   $79        Zeiger auf Befehl, im dem der Fehler auftrat
F2F9-   STA   $DC        für RESUME retten
F2FB-   LDA   $7A
F2FD-   STA   $DD
F2FF-   LDA   $F4        Zeiger auf ONERR-Statement
F301-   STA   $B8        in CHRGET-Zeiger übertragen
F303-   LDA   $F5
F305-   STA   $B9
F307-   LDA   $F6        Zeilennummer des ONERR-Statements
```

```
F309-   STA   $75        als aktuelle Zeilennummer übernehmen
F30B-   LDA   $F7
F30D-   STA   $76
F30F-   JSR   $00B7      Zeichen holen und
F312-   JSR   $D93E      GOTO-Routine durchführen
F315-   JMP   $D7D2      weiter im Programm mit Fehlerbehandlung
---------------------
```

Applesoft-Routine RESUME

```
F318-   LDA   $DA        Zeilennummer, in der der Fehler auftrat
F31A-   STA   $75        wieder als aktuelle Zeilennummer übernehmen
F31C-   LDA   $DB
F31E-   STA   $76
F320-   LDA   $DC        Zeiger auf den Befehl, in dem der Fehler auftrat
F322-   STA   $B8        wieder als CHRGET-Zeiger übernehmen
F324-   LDA   $DD
F326-   STA   $B9
F328-   LDX   $DF        geretteten Stackpointer
F32A-   TXS              wieder einsetzen
F32B-   JMP   $D7D2      weiter im Programm mit dem unterbrochenen Befehl
---------------------
F32E-   JMP   $DEC9      SYNTAX ERROR ausgeben
---------------------
```

Applesoft-Routine DEL

```
F331-   BCS   $F32E      erstes Zeichen hinter DEL keine Ziffer -> SYNTAX ERROR
F333-   LDX   $AF        Programmtext-Ende +1
F335-   STX   $69        als Anfang der einfachen Variablen übernehmen
F337-   LDX   $B0
F339-   STX   $6A
F33B-   JSR   $DA0C      Zeilennummer aus Programmtext holen --> 50,51
F33E-   JSR   $D61A      angegebene Zeile suchen
F341-   LDA   $9B        Zeiger auf angegebene Zeile  (vgl. $D61A)
F343-   STA   $60        retten
F345-   LDA   $9C
F347-   STA   $61
F349-   LDA   #$2C       Ascii für Komma
F34B-   JSR   $DEC0      muß folgen, sonst SYNTAX ERROR
F34E-   JSR   $DA0C      2. Zeilennummer aus Programmtext holen --> 50,51
F351-   INC   $50        erhöhen, damit angegebene Zeile noch gelöscht wird
F353-   BNE   $F357
F355-   INC   $51
F357-   JSR   $D61A      Zeile hinter der angegebenen Zeile suchen
F35A-   LDA   $9B        Zeiger auf diese Zeile
F35C-   CMP   $60        mit Zeiger auf die erste zu löschende Zeile vergleichen
F35E-   LDA   $9C
F360-   SBC   $61
F362-   BCS   $F365      liegt hinter dieser, angegebenen Bereich löschen
F364-   RTS              sonst gesamten Befehl ignorieren
```

angegebenen Bereich löschen

```
F365-   LDY   #$00
F367-   LDA   ($9B),Y    ein Zeichen von hinten holen
F369-   STA   ($60),Y    nach vorne übertragen (überschreibt Löschbereich)
F36B-   INC   $9B        Zeiger auf Quellbereich erhöhen
F36D-   BNE   $F371
F36F-   INC   $9C
F371-   INC   $60        Zeiger auf Zielbereich erhöhen
F373-   BNE   $F377
F375-   INC   $61
F377-   LDA   $69        ursprüngliches Programmtext-Ende +1
```

```
F379-   CMP   $9B        mit Zeiger auf Quellbereich vergleichen
F37B-   LDA   $6A
F37D-   SBC   $9C
F37F-   BCS   $F367      Ende noch nicht erreicht, weiter verschieben
F381-   LDX   $61        neues Programmtext-Ende +2
F383-   LDY   $60
F385-   BNE   $F388      um 1 verringern
F387-   DEX
F388-   DEY
F389-   STX   $6A        neues Programmtext-Ende +1 eintragen
F38B-   STY   $69
F38D-   JMP   $D4F2      CLEAR durchführen, Linkadressen berechen, Warmstart
--------------------
```

Applesoft-Routine GR

```
F390-   LDA   $C056      setze LORES
F393-   LDA   $C053      setze MIX
F396-   JMP   $FB40      zur Monitor-Routine  SETGR
--------------------
```

Applesoft-Routine TEXT

```
F399-   LDA   $C054      setze PAGE1
F39C-   JMP   $FB39      zur Monitor-Routine  SETTEXT
--------------------
```

Applesoft-Routine STORE

```
F39F-   JSR   $F7D9      angegebene Feldvariable suchen, Monitor-Register setzen
F3A2-   LDY   #$03       Zeiger in Feldvariable  (9B,9C zeigt auf Variablenkopf)
F3A4-   LDA   ($9B),Y    Länge der Feldvariable holen
F3A6-   TAX              HByte
F3A7-   DEY
F3A8-   LDA   ($9B),Y    LByte
F3AA-   SBC   #$01       abzüglich 1
F3AC-   BCS   $F3AF
F3AE-   DEX
F3AF-   STA   $50        in 50,51 eintragen
F3B1-   STX   $51
F3B3-   JSR   $FECD      Monitor-Routine WRITE speichert Feldlänge aufs Band
F3B6-   JSR   $F7BC      Monitor-Register mit Feldanfang / Feldende setzen
F3B9-   JMP   $FECD      Monitor-Routine WRITE speichert Feldvariable aufs Band
--------------------
```

Applesoft-Routine RECALL

```
F3BC-   JSR   $F7D9      angegebene Feldvariable suchen, Monitor-Register setzen
F3BF-   JSR   $FEFD      Monitor-Routine READ liest Feldlänge nach 50,51
F3C2-   LDY   #$02
F3C4-   LDA   ($9B),Y    Länge der angegebenen Feldvariable
F3C6-   CMP   $50        mit der auf Band gefundenen vergleichen
F3C8-   INY
F3C9-   LDA   ($9B),Y    Hbytes vergleichen
F3CB-   SBC   $51
F3CD-   BCS   $F3D2      dimensionierte Feldlänge >= Datenlänge auf Band
F3CF-   JMP   $D410      sonst OUT OF MEMORY ERROR
F3D2-   JSR   $F7BC      Monitor-Register mit Feldanfang / Feldende setzen
F3D5-   JMP   $FEFD      Monitor-Routine READ liest Feldvariable vom Band
--------------------
```

ROUTINEN FÜR HOCHAUFLÖSENDE GRAFIK

Applesoft-Routine HGR2

```
F3D8-   BIT   $C055      setze PAGE2
F3DB-   BIT   $C052      setze NOMIX
F3DE-   LDA   #$40       Flag für Page 2 setzen
F3E0-   BNE   $F3EA      weiter bei HGR

Applesoft-Routine  HGR
F3E2-   LDA   #$20       Flag für Page 1 setzen
F3E4-   BIT   $C054      setze PAGE1
F3E7-   BIT   $C053      setze MIX
F3EA-   STA   $E6        Flag  für HiRes-Page eintragen
F3EC-   LDA   $C057      setze HIRES
F3EF-   LDA   $C050      setze GRAPHICS
F3F2-   LDA   #$00       Farbmaske für schwarz

aktuelle HiRes-Page einfärben
F3F4-   STA   $1C        Farbmaske merken
F3F6-   LDA   $E6        Flag für aktuelle HiRes-Page
F3F8-   STA   $1B        als Adresse für Page-Anfang, HByte
F3FA-   LDY   #$00       LByte
F3FC-   STY   $1A
F3FE-   LDA   $1C        Farbmaske
F400-   STA   ($1A),Y    in Speicherzelle eintragen
F402-   JSR   $F47E      Maske invertieren, falls alternierende Bitfolge
F405-   INY              Zeiger erhöhen
F406-   BNE   $F3FE
F408-   INC   $1B
F40A-   LDA   $1B
F40C-   AND   #$1F       ergibt 0, falls Page-Ende erreicht
F40E-   BNE   $F3FE      Page noch weiter füllen
F410-   RTS
--------------------

RAM-Adresse aus XY-Koordinaten berechnen
I: X-Koordinate in X,Y;  Y-Koordinate im Accu
O: RAM-Adresse des Zeilenanfangs in 26,27
   Position in der Zeile in E5 = Y;  Bitmaske in 30
F411-   STA   $E2        Y-Koordinate merken
F413-   STX   $E0        X-Koordinate merken
F415-   STY   $E1                                  Block  Textzeile  Grafikzeile
F417-   PHA              Bezeichnung von Bit7-0:   B1,B0, T2,T1,T0,  G2,G1,G0
F418-   AND   #$C0       Bits 6 und 7 herausholen
F41A-   STA   $26        ergibt Blocknummer x 64  (Block bedeutet Page-Drittel)
F41C-   LSR
F41D-   LSR              ergibt Blocknummer x 16
F41E-   ORA   $26        plus Blocknummer x 64  ergibt  Blocknummer x 80
F420-   STA   $26        eintragen
F422-   PLA              Y-Koordinate (vgl. Info bei $F417)
F423-   STA   $27        eintragen
F425-   ASL
F426-   ASL              damit Textzeile linksbündig (T2 in Bit 7)
F427-   ASL              T2 --> C
F428-   ROL   $27        C --> Bit0 von 27, übrige Bits hochschieben
F42A-   ASL              T1 --> C
F42B-   ROL   $27        C --> Bit0 von 27, übrige Bits hochschieben
F42D-   ASL              T0 --> C
F42E-   ROR   $26        C --> Bit7 ergibt in 26:   T0 (Block x40) 0  0  0
F430-   LDA   $27        momentaner Inhalt in 27:   T2 T1 T0 G2 G1 G0 T2 T1
F432-   AND   #$1F       lösche Bits 5-7
F434-   ORA   $E6        setze Page-Flag dafür ein
F436-   STA   $27        ergibt:                    T0 Page  G2 G1 G0 T2 T1
F438-   TXA              X-Koordinate LByte
```

```
F439-   CPY   #$00       HByte mit 0 vergleichen
F43B-   BEQ   $F442      X-Koordinate < 256
F43D-   LDY   #$23       sonst Position mit 35 vorbesetzen (d.h. 36. Spalte)
F43F-   ADC   #$04       LByte korrigieren, da 36 x 7 = 252 = 256 - 4
F441-   INY
F442-   SBC   #$07       Division durch 7, Y als Zähler
F444-   BCS   $F441      weiter
F446-   STY   $E5        Spalte (d.h. Byteposition) eintragen
F448-   TAX              = Bitposition im letzen Byte -7  (256 - 7 = 249)
F449-   LDA   $F4B9,X    d.h. $F5B2 + Bitposition, ergibt Bitmaske
F44C-   STA   $30        eintragen
F44E-   TYA              Spalte
F44F-   LSR              Bit0 --> C, gesetzt falls ungerade Spalte
F450-   LDA   $E4        Farbmaske
F452-   STA   $1C        übertragen
F454-   BCS   $F47E      ungerade Spalte, Maske invertieren falls alternierend
F456-   RTS
--------------------

einzelnen Punkt setzen
I: Koordinaten in A,X,Y
F457-   JSR   $F411      RAM-Adresse berechnen
F45A-   LDA   $1C        Farbmaske
F45C-   EOR   ($26),Y    mit Speicherzelle verknüpfen
F45E-   AND   $30        Bitmaske holt Bit heraus
F460-   EOR   ($26),Y    damit Bit aus Farbmaske kopiert
F462-   STA   ($26),Y    in Speicherzelle eintragen
F464-   RTS
--------------------

RAM-Adresse für linken oder rechten Nachbarpunkt
F465-   BPL   $F48A      Bit7=0, Adresse für rechten Nachbarpunkt berechnen

linker Nachbarpunkt
F467-   LDA   $30        Bitmaske, Bit7=1
F469-   LSR              nach links schieben  (LSR wegen gespiegelter Darst.)
F46A-   BCS   $F471      war Randbit, neuer Punkt im nächsten Byte
F46C-   EOR   #$C0       Bit7 setzen, Bit6 löschen
F46E-   STA   $30        neue Bitmaske eintragen
F470-   RTS
F471-   DEY              Spaltenindex auf linke Nachbarspalte
F472-   BPL   $F476      linker Bildschirmrand nicht erreicht, ok.
F474-   LDY   #$27       sonst weiter in Spalte 39 (rechter Bildschirmrand)
F476-   LDA   #$C0       Bitmaske für Bit6, Bit7=1 wie immer
F478-   STA   $30        neue Bitmaske eintragen
F47A-   STY   $E5        neuen Spaltenindex eintragen

Farbmaske invertieren, falls alternierende Bitfolge
F47C-   LDA   $1C        Farbmaske
F47E-   ASL              verschieben
F47F-   CMP   #$C0       Bits 5 und 6 prüfen
F481-   BPL   $F489      Bit5 = Bit6, alterniert nicht --> nicht invertieren
F483-   LDA   $1C        sonst Farbmaske
F485-   EOR   #$7F       invertieren
F487-   STA   $1C        neue Farbmaske eintragen
F489-   RTS

rechter Nachbarpunkt
F48A-   LDA   $30        Bitmaske
F48C-   ASL              nach rechts schieben (ASL wegen gespiegelter Darst.)
F48D-   EOR   #$80       Bit7 (war vorher Bit6) invertieren
```

```
F48F-   BMI   $F46E      Bit6 war 0, neuer Punkt im selben Byte, fertig
F491-   LDA   #$81       Bitmaske für Bit0, Bit7=1 wie immer
F493-   INY              Spaltenindex auf rechte Nachbarspalte setzen
F494-   CPY   #$28       rechter Bildschirmrand erreicht?
F496-   BCC   $F478      nein, ok.
F498-   LDY   #$00       sonst neuer Spaltenindex 0 (linker Bildschirmrand)
F49A-   BCS   $F478      Routine abschließen
--------------------

Punkt invertieren für  XDRAW
F49C-   CLC
F49D-   LDA   $D1        Bits 0,1,2 = aktuelle SHAPE-Anweisung
F49F-   AND   #$04       Bit2 = Plot-Flag
F4A1-   BEQ   $F4C8      Punkt nicht verändern, neuen Punkt bestimmen
F4A3-   LDA   #$7F
F4A5-   AND   $30        Bitmaske, Bit7=0
F4A7-   AND   ($26),Y    Originalbit holen
F4A9-   BNE   $F4C4      Bit =1, wird nullgesetzt
F4AB-   INC   $EA        sonst Collision Counter erhöhen
F4AD-   LDA   #$7F
F4AF-   AND   $30        Bitmaske um Punkt zu setzen
F4B1-   BPL   $F4C4      Byte eintragen und neuen Punkt bestimmen

Punkt setzen für  DRAW
F4B3-   CLC
F4B4-   LDA   $D1        Bits 0,1,2 = aktuelle SHAPE-Anweisung
F4B6-   AND   #$04       Bit2 = Plot-Flag
F4B8-   BEQ   $F4C8      Punkt nicht setzen, neuen Punkt bestimmen
F4BA-   LDA   ($26),Y    Originalbit
F4BC-   EOR   $1C        mit Farbmaske verknüpfen
F4BE-   AND   $30        Bit herausholen
F4C0-   BNE   $F4C4      Punkt ist noch nicht gesetzt
F4C2-   INC   $EA        sonst Collision Counter erhöhen
F4C4-   EOR   ($26),Y    neues Bit einfügen
F4C6-   STA   ($26),Y    Byte eintragen

neuen Punkt bestimmen nach SHAPE-Anweisung
F4C8-   LDA   $D1        Bits 0 und 1 enthalten Zugrichtung
F4CA-   ADC   $D3        + Quadrant entsprechend ROT-Wert und Carry-Flag
F4CC-   AND   #$03       ergibt tatsächliche Zugrichtung
F4CE-   CMP   #$02       Bit1 --> C
F4D0-   ROR              C --> Bit7;  Bit0 --> C
F4D1-   BCS   $F465      Bit0 war 1, d.h. Zugrichtung links oder rechts
--------------------

Nachbarpunkt über oder unter altem Punkt bestimmen
F4D3-   BMI   $F505      Bit7=1, d.h. RAM-Adresse für Punkt unterhalb bestimmen

Punkt oberhalb bestimmen
F4D5-   CLC
F4D6-   LDA   $27        Adresse des Zeilenanfangs, HByte
F4D8-   BIT   $F5B9      enthält 1C, d.h. Nummer der Grafikzeile herausholen
F4DB-   BNE   $F4FF      nicht oberste Grafikzeile innerhalb der Textzeile, ok
F4DD-   ASL   $26        sonst T0 --> C
F4DF-   BCS   $F4FB      ungerade Textzeile, zu HByte der Adresse 1C addieren
F4E1-   BIT   $F4CD      enthält 03, d.h. Nummer der geraden Textzeile holen
F4E4-   BEQ   $F4EB      oberste Textzeile im Block, weiter im Block oberhalb
F4E6-   ADC   #$1F       Textzeile 2,4 oder 6, zu HByte der Adresse 1B addieren
F4E8-   SEC
F4E9-   BCS   $F4FD      Addition vervollständigen
F4EB-   ADC   #$23       zu 20 oder 40 (je nach Page) addieren, ergibt 43/63
```

```
F4ED-   PHA                 merken

neuer Block
F4EE-   LDA   $26           00        /  50        /  A0          für Block 0/1/2
F4F0-   ADC   #$B0          B0, C=0   /  00, C=1   /  50, C=1
F4F2-   BCS   $F4F6
F4F4-   ADC   #$F0
F4F6-   STA   $26           A0, C=1   /  00, C=1   /  50, C=1  (später 50/00/28)
F4F8-   PLA                 43 (63) minus 4 ergibt 3F (5F), d.h. unterste Zeile
F4F9-   BCS   $F4FD         Subtraktion durchführen, Routine abschließen
F4FB-   ADC   #$1F          C=1, also 20 addieren
F4FD-   ROR   $26           ASL rückgängig machen, C --> Bit7
F4FF-   ADC   #$FC          minus 4
F501-   STA   $27           HByte eintragen
F503-   RTS
--------------------

Punkt unterhalb bestimmen
F504-   CLC
F505-   LDA   $27           Adresse des Zeilenanfangs, HByte
F507-   ADC   #$04          plus 4 für Grafikzeile darunter
F509-   BIT   $F5B9         enthält 1C, d.h. Nummer der Grafikzeile herausholen
F50C-   BNE   $F501         war nicht unterste Grafikzeile der Textzeile, ok
F50E-   ASL   $26           sonst T0 --> C
F510-   BCC   $F52A         gerade Textzeile, 20 abziehen und Routine abschließen
F512-   ADC   #$E0          minus 1F ergibt 21/22/23/24 bei Textzeile 1/3/5/7
F514-   CLC
F515-   BIT   $F508         enthält 04, d.h. Bit 2 herausholen
F518-   BEQ   $F52C         nicht unterste Textzeile im Block, ok.

neuer Block
F51A-   LDA   $26           00        /  50        /  A0          bei Block 0/1/2
F51C-   ADC   #$50          50, C=0   /  A0, C=0   /  F0, C=0
F51E-   EOR   #$F0          A0        /  50        /  00
F520-   BEQ   $F524
F522-   EOR   #$F0
F524-   STA   $26           50        /  A0        /  00
F526-   LDA   $E6           20 bzw. 40, je nach Page
F528-   BCC   $F52C
F52A-   ADC   #$E0          20 subtrahieren
F52C-   ROR   $26           ASL rückgängig machen, C --> Bit7
F52E-   BCC   $F501         Routine abschließen
--------------------

alte Koordinaten löschen
F530-   PHA                 Accu retten
F531-   LDA   #$00
F533-   STA   $E0           Koordinaten löschen
F535-   STA   $E1
F537-   STA   $E2
F539-   PLA                 Accu wieder holen
--------------------

Linie zeichnen
I: alte Koordinaten (E0,E1 / E2), neue (A,X / Y)

Koordinatendifferenzen bestimmen
F53A-   PHA                 neue X-Koordinate, LByte retten
F53B-   SEC
F53C-   SBC   $E0           minus alte X-Koordinate, LByte
F53E-   PHA                 X-Differenz, LByte retten
```

```
F53F-   TXA                 neue X-Koordinate, HByte
F540-   SBC   $E1           minus alte X-Koordinate, HByte
F542-   STA   $D3           X-Differenz, HByte retten    (= FF, 0 oder 1)
F544-   BCS   $F550         neue X-Koordinate  >=  alte X-Koordinate
F546-   PLA                 sonst Betrag der X-Differenz bestimmen
F547-   EOR   #$FF
F549-   ADC   #$01          Betrag der X-Differenz, LByte (mit dXL bezeichnet)
F54B-   PHA                 wieder retten
F54C-   LDA   #$00
F54E-   SBC   $D3           Betrag der X-Differenz, HByte (mit dXH bezeichnet)
F550-   STA   $D1           ins dXH-Register
F552-   STA   $D5           und ins Saldo-Register, HByte
F554-   PLA                 dXL
F555-   STA   $D0           ins dXL-Register
F557-   STA   $D4           und ins Saldo-Register, LByte
F559-   PLA                 neue X-Koordinate, LByte
F55A-   STA   $E0           übernehmen
F55C-   STX   $E1           HByte übernehmen
F55E-   TYA                 neue Y-Koordinate
F55F-   CLC
F560-   SBC   $E2           minus alte Y-Koordinate
F562-   BCC   $F568         neue Y-Koordinate  =<  alte Y-Koordinate
F564-   EOR   #$FF          sonst negieren
F566-   ADC   #$FE
F568-   STA   $D2           -1 - Betrag der Y-Differenz (mit  -dY-1 bezeichnet)
F56A-   STY   $E2           neue Y-Koordinate übernehmen

Richtungsflag und Schrittzähler setzen
F56C-   ROR   $D3           C --> Bit7, im Richtungsflag also: Bit7=1  nach oben
F56E-   SEC                                                    Bit7=0  nach unten
F56F-   SBC   $D0           - dXL = -dY -dXL -1                Bit6=1  nach links
F571-   TAX                 als Schrittzähler LByte            Bit6=0  nach rechts
F572-   LDA   #$FF          -1
F574-   SBC   $D1           minus dXH ergibt -dXH - 1
F576-   STA   $1D           als Schrittzähler HByte
F578-   LDY   $E5           Spaltenindex nach Y
F57A-   BCS   $F581         nun die Linie zeichnen

Linie durch Treppe annähern

Bewegung in X-Richtung
F57C-   ASL                 Bit6 --> Bit7 (nach links oder nach rechts)
F57D-   JSR   $F465         Adresse für neuen Punkt bestimmen
F580-   SEC
F581-   LDA   $D4           Saldo - dY -1 +1    (+1 wegen C=1)
F583-   ADC   $D2
F585-   STA   $D4           neuer Saldo, LByte
F587-   LDA   $D5
F589-   SBC   #$00
F58B-   STA   $D5           HByte

Punkt auf Bildschirm bringen
F58D-   LDA   ($26),Y       Speicherzelle holen
F58F-   EOR   $1C           mit Farbmaske verknüpfen
F591-   AND   $30           entscheidendes Bit herausholen
F593-   EOR   ($26),Y       und einfügen
F595-   STA   ($26),Y       Byte abspeichern
F597-   INX                 Schrittzähler dekrementieren  (negativ!)
F598-   BNE   $F59E
F59A-   INC   $1D           HByte dekrementieren
F59C-   BEQ   $F600         Endpunkt erreicht, fertig
```

```
F59E-   LDA   $D3        Richtungsflag
F5A0-   BCS   $F57C      Saldo positiv, deswegen Bewegung in X-Richtung

Bewegung in Y-Richtung
F5A2-   JSR   $F4D3      Adresse für neuen Punkt bestimmen (nach Bit7 vom Flag)
F5A5-   CLC
F5A6-   LDA   $D4        Saldo + Betrag der X-Differenz
F5A8-   ADC   $D0
F5AA-   STA   $D4        neuer Saldo, LByte
F5AC-   LDA   $D5
F5AE-   ADC   $D1
F5B0-   BVC   $F58B      HByte eintragen, Punkt auf Bildschirm bringen
---------------------
F5B2-   81 82 84 88 90 A0 C0     Bitmasken
---------------------
F5B9-   1C                       Hilfsbyte für BIT
---------------------

Tabelle der Cosinuswerte für ROT
F5BA-   FF FE FA F4 EC E1 D4 C5 B4    0....Pi/4
F5C3-   A1 8D 78 61 49 31 18           ....Pi/2
F5CA-   FF = -1                       Pi/2
---------------------

XY-Koordinaten aus RAM-Adresse berechnen
I: 26,27 Zeilenadresse; E5 Spaltenindex; 30 Bitmaske
O: E0,E1 X-Koordinate; E2 Y-Koordinate
F5CB-   LDA   $26        enthält als einzelne Bits     T0  (Blockx40)  0  0  0
F5CD-   ASL              T0 --> C                   / Bit7  6  5  4  3  2  1  0
F5CE-   LDA   $27        enthält als einzelne Bits      0  Page  G2 G1 G0 T2 T1
F5D0-   AND   #$03       T2 und T1 herausholen
F5D2-   ROL              T0 hinten anfügen
F5D3-   ORA   $26        an Blockbits hinten anfügen
F5D5-   ASL              um 3 Bit hochschieben
F5D6-   ASL
F5D7-   ASL              ergibt                        B1 B0 T2 T1 T0  0  0  0
F5D8-   STA   $E2        als Y-Koordinate
F5DA-   LDA   $27
F5DC-   LSR              um 2 Bit nach rechts schieben
F5DD-   LSR
F5DE-   AND   #$07       Bits 0,1,2 herausholen (= G2,G1,G0)
F5E0-   ORA   $E2        in Y-Koordinate einfügen
F5E2-   STA   $E2
F5E4-   LDA   $E5        Spaltenindex
F5E6-   ASL              x 2
F5E7-   ADC   $E5        plus Spaltenindex ergibt 3-fachen Wert
F5E9-   ASL              x 2
F5EA-   TAX              ergibt 6-fachen Wert
F5EB-   DEX
F5EC-   LDA   $30        Bitmaske
F5EE-   AND   #$7F       Bit7 löschen
F5F0-   INX              X zählt mit
F5F1-   LSR              verschieben
F5F2-   BNE   $F5F0      gesetztes Bit noch im Byte, weiterschieben
F5F4-   STA   $E1        X-Koordinate, HByte zunächst nullsetzen
F5F6-   TXA              Bitspalte +  6 x Spaltenindex
F5F7-   CLC
F5F8-   ADC   $E5        plus Spaltenindex ergibt X-Koordinate, LByte
F5FA-   BCC   $F5FE
F5FC-   INC   $E1        HByte auf 1 erhöhen, falls X-Koordinate > 255
F5FE-   STA   $E0        LByte eintragen
```

```
F600-   RTS
--------------------

DRAW-Routine
I: X,Y Zeiger auf SHAPE-Definition, Accu = ROT-Wert
F601-   STX   $1A        Zeiger eintragen
F603-   STY   $1B

Haupteinsprung
F605-   TAX               ROT-Wert retten
F606-   LSR               Bits 4-7 --> Bits 0-3
F607-   LSR
F608-   LSR
F609-   LSR
F60A-   STA   $D3         als Quadrant (Anzahl rechter Winkel im Uhrzeigersinn)
F60C-   TXA               Rot-Wert
F60D-   AND   #$0F        Bits 0-3 herausholen
F60F-   TAX               ergibt Winkel innerhalb des Quadranten
F610-   LDY   $F5BA,X     zugehörigen Cosinuswert aus Tabelle holen
F613-   STY   $D0         und retten
F615-   EOR   #$0F        ergibt 15 - Tabellenindex
F617-   TAX
F618-   LDY   $F5BB,X     Tabellenwert von 16 - Index  ($F5BB!)
F61B-   INY               ergibt Sinuswert
F61C-   STY   $D2         merken
F61E-   LDY   $E5         Spaltenindex holen
F620-   LDX   #$00
F622-   STX   $EA         Collision Counter nullsetzen
F624-   LDA   ($1A,X)     hole 1. Byte der Shape-Definition

Shape-Anweisung durchführen
F626-   STA   $D1         Bits 0,1,2 enthält Shape-Anweisung
F628-   LDX   #$80
F62A-   STX   $D4         Saldo-Register für Cosinus
F62C-   STX   $D5         und für Sinus initialisieren
F62E-   LDX   $E7         SCALE-Wert --> X als Zähler

Verhältnis des Sin-/Cos-Werts bestimmt Linienrichtung
F630-   LDA   $D4         Cosinus-Saldo
F632-   SEC
F633-   ADC   $D0         um Cosinus-Wert erhöhen
F635-   STA   $D4
F637-   BCC   $F63D       kein Überlauf
F639-   JSR   $F4B3       Punkt setzen, nächsten Punkt in Richtung der Shape-
F63C-   CLC               Anweisung plus Quadrant bestimmen (= Nullrichtung)
F63D-   LDA   $D5         Sinus-Saldo
F63F-   ADC   $D2         um Sinus-Wert erhöhen
F641-   STA   $D5
F643-   BCC   $F648       kein Überlauf
F645-   JSR   $F4B4       Punkt setzen, nächsten in Nullrichtung + 90' bestimmen
F648-   DEX               Zähler dekrementieren                  ($F4B4,C=1!)
F649-   BNE   $F630       weiter mit dieser Shape-Anweisung
F64B-   LDA   $D1         Byte mit Shape-Anweisungen
F64D-   LSR               nächste Anweisung in Bits 3,4,5 --> Bits 0,1,2
F64E-   LSR
F64F-   LSR
F650-   BNE   $F626       Byte nicht 0, d.h. gültige Shape-Anweisung in Bit 0,1,2
F652-   INC   $1A         sonst Zeiger auf Shape-Definition erhöhen
F654-   BNE   $F658
F656-   INC   $1B
F658-   LDA   ($1A,X)     und neues Byte mit Shape-Anweisungen holen
```

```
F65A-   BNE   $F626       ungleich 0, d.h. Endbyte noch nicht erreicht
F65C-   RTS               sonst fertig
--------------------

XDRAW-Routine
I: X,Y Zeiger auf Shape-Definition, Accu = ROT-Wert
F65D-   STX   $1A         Zeiger eintragen
F65F-   STY   $1B

Haupteinsprung
F661-   TAX               ROT-Wert retten
F662-   LSR               Bits 4-7 --> Bits 0-3
F663-   LSR
F664-   LSR
F665-   LSR
F666-   STA   $D3         als Quadrant (Anzahl rechter Winkel im Uhrzeigersinn)
F668-   TXA               ROT-Wert
F669-   AND   #$0F        Bits 0-3 herausholen
F66B-   TAX               ergibt Winkel innerhalb des Quadranten
F66C-   LDY   $F5BA,X     zugehörigen Cosinus-Wert aus Tabelle holen
F66F-   STY   $D0         und merken
F671-   EOR   #$0F        ergibt 15 - Tabellenindex
F673-   TAX
F674-   LDY   $F5BB,X     Tabellenwert von 16 - Index  ($F5BB!)
F677-   INY               ergibt Sinus-Wert
F678-   STY   $D2         merken
F67A-   LDY   $E5         Spaltenindex holen
F67C-   LDX   #$00
F67E-   STX   $EA         Collision Counter nullsetzen
F680-   LDA   ($1A,X)     hole 1. Byte der Shape-Definition

Shape-Anweisung durchführen
F682-   STA   $D1         Bits 0,1,2 enthalten Shape-Anweisung
F684-   LDX   #$80
F686-   STX   $D4         Saldo-Register für Cosinus
F688-   STX   $D5         und für Sinus initialisieren
F68A-   LDX   $E7         SCALE-Wert --> X als Zähler

Verhältnis des Sin-/Cos-Werts bestimmt Linienrichtung
F68C-   LDA   $D4         Cosinus-Saldo
F68E-   SEC
F68F-   ADC   $D0         um Cosinus-Wert erhöhen
F691-   STA   $D4
F693-   BCC   $F699       kein Überlauf
F695-   JSR   $F49C       Punkt invertieren, nächsten  in Richtung der Shape-
F698-   CLC               Anweisung plus Quadrant bestimmen (= Nullrichtung)
F699-   LDA   $D5         Sinus-Saldo
F69B-   ADC   $D2         um Sinus-Wert erhöhen
F69D-   STA   $D5
F69F-   BCC   $F6A4       kein Überlauf
F6A1-   JSR   $F49D       Punkt invert., nächsten in Nullrichtung + 90' bestimmen
F6A4-   DEX               Zähler dekrementieren                  ($F49D,C=1!)
F6A5-   BNE   $F68C       weiter mit dieser Shape-Anweisung
F6A7-   LDA   $D1         Byte mit Shape-Anweisungen
F6A9-   LSR               nächste Anweisung in Bits 3,4,5 --> Bits 0,1,2
F6AA-   LSR
F6AB-   LSR
F6AC-   BNE   $F682       Byte nicht 0, d.h. gültige Shape-Anweisung in Bit 0,1,2
F6AE-   INC   $1A         sonst Zeiger auf Shape-Definition erhöhen
F6B0-   BNE   $F6B4
F6B2-   INC   $1B
```

```
F6B4-   LDA   ($1A,X)    und neues Byte mit Shape-Anweisungen holen
F6B6-   BNE   $F682      ungleich 0, d.h. Endbyte noch nicht erreicht
F6B8-   RTS              sonst fertig
--------------------

Parameter für HPLOT, DRAW, XDRAW holen
O: X-Koordinate in X,Y;  Y-Koordinate in A = 9D
F6B9-   JSR   $DD67      numerischen Ausdruck auswerten --> FAC1
F6BC-   JSR   $E752      FAC1 in Integer (in 50,51) umwandeln (= X-Koordinate)
F6BF-   LDY   $51        und nach X,Y bringen
F6C1-   LDX   $50
F6C3-   CPY   #$01       HByte mit 1 vergleichen
F6C5-   BCC   $F6CD      ok, X-Koordinate < 256
F6C7-   BNE   $F6E6      X-Koordinate > 511, --> ILLEGAL QUANTITY ERROR
F6C9-   CPX   #$18       LByte mit 24 vergleichen
F6CB-   BCS   $F6E6      X-Koordinate > 279, --> ILLEGAL QUANTITY ERROR
F6CD-   TXA              sonst X-Koordinate auf Stack retten
F6CE-   PHA
F6CF-   TYA
F6D0-   PHA
F6D1-   LDA   #$2C       Ascii für Komma
F6D3-   JSR   $DEC0      muß folgen, sonst SYNTAX ERROR
F6D6-   JSR   $E6F8      hole 1-Byte-Integer aus Programmtext --> X
F6D9-   CPX   #$C0       Y-Koordinate mit 192 vergleichen
F6DB-   BCS   $F6E6      Y-Koordinate > 191, --> ILLEGAL QUANTITY ERROR
F6DD-   STX   $9D        sonst Y-Koordinate nach 9D
F6DF-   PLA              und X-Koordinate vom Stack nach X,Y bringen
F6E0-   TAY
F6E1-   PLA
F6E2-   TAX
F6E3-   LDA   $9D        Y-Koordinate nach Accu,
F6E5-   RTS              fertig
--------------------
F6E6-   JMP   $F206      ILLEGAL QUANTITY ERROR ausgeben
--------------------

Applesoft-Routine  HCOLOR
F6E9-   JSR   $E6F8      hole 1-Byte-Integer aus Programmtext --> X
F6EC-   CPX   #$08       Farbcode mit 8 vergleichen
F6EE-   BCS   $F6E6      Farbcode > 7, --> ILLEGAL QUANTITY ERROR
F6F0-   LDA   $F6F6,X    Farbmaske aus Tabelle holen
F6F3-   STA   $E4        und eintragen
F6F5-   RTS
--------------------

Tabelle der Farbmasken
F6F6-   00 2A          00000000   00101010
F6F8-   55 7F          01010101   01111111
F6FA-   80 AA          10000000   10101010
F6FC-   D5 FF          11010101   11111111
--------------------

Applesoft-Routine  HPLOT
F6FE-   CMP   #$C1       folgt Token für TO?
F700-   BEQ   $F70F      nein, weiter
F702-   JSR   $F6B9      sonst Punkt-Koordinaten holen --> X,Y / A
F705-   JSR   $F457      RAM-Adresse berechnen, Punkt setzen
F708-   JSR   $00B7      hole Zeichen aus Programmtext
F70B-   CMP   #$C1       Token für TO?
F70D-   BNE   $F6F5      nein, fertig
F70F-   JSR   $DEC0      sonst neues Zeichen holen
```

```
F712-   JSR   $F6B9      Punkt-Koordinaten holen --> X,Y / A
F715-   STY   $9D        X-Koordinate nach A,X und Y-Koordinate nach Y bringen
F717-   TAY
F718-   TXA
F719-   LDX   $9D
F71B-   JSR   $F53A      Linie zeichnen
F71E-   JMP   $F708      weiter mit nächster Linie oder fertig
--------------------

Applesoft-Routine  ROT=
F721-   JSR   $E6F8      hole 1-Byte-Integer aus Programmtext --> X
F724-   STX   $F9        als ROT-Wert abspeichern
F726-   RTS
--------------------

Applesoft-Routine  SCALE=
F727-   JSR   $E6F8      hole 1-Byte-Integer aus Programmtext --> X
F72A-   STX   $E7        als SCALE-Wert abspeichern
F72C-   RTS
--------------------

hole Parameter für  DRAW, XDRAW
O: 1A,1B zeigt auf Shape-Definition, ROT-Wert im Accu
F72D-   JSR   $E6F8      1-Byte-Integer aus Programmtext holen --> X
F730-   LDA   $E8        Zeiger auf Kopf des Shape-Tables
F732-   STA   $1A        nach 1A,1B bringen
F734-   LDA   $E9
F736-   STA   $1B
F738-   TXA              Shape-Nummer
F739-   LDX   #$00
F73B-   CMP   ($1A,X)    mit Anzahl der Shape-Definitionen im Table vergleichen
F73D-   BEQ   $F741      gleich, ok
F73F-   BCS   $F6E6      zu groß, --> ILLEGAL QUANTITY ERROR
F741-   ASL              verdoppeln
F742-   BCC   $F747      Shape-Nummer < 128
F744-   INC   $1B        sonst Zeiger um 256 erhöhen
F746-   CLC
F747-   TAY              damit (1A),Y Zeiger auf Abstands-Tabelle
F748-   LDA   ($1A),Y    Abstand der Shape-Definition vom Anfang des Shapetable
F74A-   ADC   $1A        zum Anfang des Shapetables addieren
F74C-   TAX
F74D-   INY
F74E-   LDA   ($1A),Y    Hbyte addieren
F750-   ADC   $E9
F752-   STA   $1B        Zeiger auf richtige Shape-Definition eintragen
F754-   STX   $1A
F756-   JSR   $00B7      hole Zeichen
F759-   CMP   #$C5       Token für AT?
F75B-   BNE   $F766      kein Startpunkt angegeben
F75D-   JSR   $DEC0      sonst neues Zeichen holen
F760-   JSR   $F6B9      Koordinaten des Startpunktes holen
F763-   JSR   $F411      RAM-Adresse für Startpunkt bestimmen
F766-   LDA   $F9        ROT-Wert laden
F768-   RTS
--------------------

Applesoft-Routine  DRAW
F769-   JSR   $F72D      DRAW-Parameter holen  (s.d.)
F76C-   JMP   $F605      DRAW-Routine ausführen
--------------------

Applesoft-Routine  XDRAW
```

```
F76F-   JSR   $F72D      DRAW-Parameter holen  (s.d.)
F772-   JMP   $F661      XDRAW-Routine ausführen
---------------------

Applesoft-Routine  SHLOAD
F775-   LDA   #$00
F777-   STA   $3D        Zeiger 3C,3D auf $0050 setzen,
F779-   STA   $3F        Zeiger 3E,3F auf $0051 setzen
F77B-   LDY   #$50
F77D-   STY   $3C
F77F-   INY
F780-   STY   $3E
F782-   JSR   $FEFD      Monitor-Routine READ liest Länge des Shapetables
F785-   CLC              vom Band nach 50,51
F786-   LDA   $73        HIMEM
F788-   TAX
F789-   DEX              minus 1
F78A-   STX   $3E        nach 3E,3F als Endadresse für Laderoutine
F78C-   SBC   $50        minus Länge des Shapetable
F78E-   PHA              retten
F78F-   LDA   $74        HBytes bestimmen
F791-   TAY
F792-   INX
F793-   BNE   $F796
F795-   DEY
F796-   STY   $3F
F798-   SBC   $51        minus Länge ergibt Zeiger auf Table-Anfang, HByte
F79A-   CMP   $6E        mit Feldvariablen-Ende +1 vergleichen
F79C-   BCC   $F7A0      Platz reicht nicht aus, SYNTAX ERROR
F79E-   BNE   $F7A3      Platz reicht aus, ok
F7A0-   JMP   $D410      SYNTAX ERROR ausgeben
F7A3-   STA   $74        Shapetable-Anfang als neues HIMEM
F7A5-   STA   $70        und als Anfang des Stringbereichs eintragen
F7A7-   STA   $3D        sowie als Anfangsadresse für die Laderoutine
F7A9-   STA   $E9        und als Zeiger auf Anfang des Shapetables
F7AB-   PLA              Shapetable-Anfang, LByte
F7AC-   STA   $E8        jeweils eintragen
F7AE-   STA   $73
F7B0-   STA   $6F
F7B2-   STA   $3C
F7B4-   JSR   $FCFA      Monitor-Routine lädt Shape-Table vom Band in den
F7B7-   LDA   #$03       angegebenen Bereich
F7B9-   JMP   $FF02
---------------------

setze Monitor-Register für STORE / RECALL
I: Länge der Feldvariablen in 50,51
   Zeiger auf Variablenkopf in 9B,9C
F7BC-   CLC
F7BD-   LDA   $9B        Zeiger auf Variablenkopf
F7BF-   ADC   $50        plus Variablenlänge
F7C1-   STA   $3E        ins Bereichsende-Register 3E,3F
F7C3-   LDA   $9C
F7C5-   ADC   $51
F7C7-   STA   $3F
F7C9-   LDY   #$04
F7CB-   LDA   ($9B),Y    Zeiger auf Anzahl der Dimensionen des Feldes
F7CD-   JSR   $E0EF      Zeiger auf 1. Datenbyte nach 94,95 bringen
F7D0-   LDA   $94        Zeiger auf 1. Datenbyte
F7D2-   STA   $3C        ins Bereichsanfang-Register 3C,3D
F7D4-   LDA   $95
```

```
F7D6-   STA   $3D
F7D8-   RTS
---------------------

suche Feld für STORE / RECALL
F7D9-   LDA   #$40
F7DB-   STA   $14        STORE-Flag setzen
F7DD-   JSR   $DFE3      angegebene Feldvariable suchen
F7E0-   LDA   #$00
F7E2-   STA   $14        STORE-Flag löschen
F7E4-   JMP   $D8F0      setze Monitor-Register für Längendaten
---------------------

Applesoft-Routine  HTAB
F7E7-   JSR   $E6F8      hole 1-Byte-Integer aus Programmtext --> X
F7EA-   DEX              1. Spalte hat Spaltennummer 0
F7EB-   TXA              horizontale Cursorposition
F7EC-   CMP   #$28       mit 40 vergleichen
F7EE-   BCC   $F7FA      ok, kleiner, als Cursor-Spalte eintragen
F7F0-   SBC   #$28       sonst 40 subtrahieren
F7F2-   PHA              Ergebnis retten
F7F3-   JSR   $DAFB      Zeilenvorschub auslösen
F7F6-   PLA              verbleibender Abstand
F7F7-   JMP   $F7EC      weitere Zeilenvorschübe oder fertig
F7FA-   STA   $24        Cursor-Spalte eintragen ( = Eingabe -1 mod 40)
F7FC-   RTS
---------------------

F7FD-   CB D2 D7         ???
---------------------
```

ab hier folgt das Monitor-ROM

Übersicht über die ROM-Routinen, alphabetisch sortiert

EBAF	ABS, Betragsfunktion
DF55	AND, logische Und-Verknüpfung
E6E5	ASC, bringt Ascii-Wert des ersten Zeichens
F09E	ATN, Arcustangens-Funktion
DD7B	Auswertung eines beliebigen Ausdrucks, Unterprogramm
DF0C	Auswertung von Funktionen
F1D5	CALL, Maschinenprogramm-Aufruf
F10B	CHRGET-Routine
D697	CHRGET-Zeiger auf Programmstart -1 setzen, Unterprogramm
E646	CHR$, wandelt Ascii-Kode in Stringzeichen um
D66A	CLEAR, löscht alle Variablen
F24F	COLOR, Farbe für LoRes-Grafik wählen
D896	CONT, unterbrochenes Programm fortführen
EFEA	COS, Cosinus-Funktion
E313	DEF, Funktion definieren
F331	DEL, Bereich in Programmtext löschen
DFD9	DIM, Feldvariable dimensionieren
F769	DRAW, Shape darstellen
F605	DRAW, Unterprogramm
DB3A	Druckroutine für Strings, Unterprogramm
DB57	Druckroutine für einzelne Zeichen, Unterprogramm
D870	END, Programmende
EF09	EXP, Exponential-Funktion
EBF2	FAC1 in Integer umwandeln, Unterprogramm
E752	FAC1 in Integer umwandeln und nach 50,51 bringen, Unterprogramm
ED34	FAC1 in Zahlenstring umwandeln, Unterprogramm
EB63	FAC1 nach FAC2 übertragen, Unterprogramm
EB2B	FAC1 nach (X,Y) übertragen, Unterprogramm
EB72	FAC1 runden, Unterprogramm
EBB2	FAC1 vergleichen mit (A,Y), Unterprogramm
EA39	FAC1 = FAC1 * 10, Unterprogramm
EA55	FAC1 = FAC1 / 10, Unterprogramm
E7A0	FAC1 = FAC1 + 0.5, Unterprogramm
E7C6	FAC1 = FAC2 + FAC1, Unterprogramm
E7AA	FAC1 = FAC2 - FAC1, Unterprogramm
E987	FAC1 = FAC2 * FAC1, Unterprogramm
EA69	FAC1 = FAC2 / FAC1, Unterprogramm
EE97	FAC1 = FAC2 ^ FAC1, Unterprogramm
E7BE	FAC1 = (A,Y) + FAC1, Unterprogramm
E7A7	FAC1 = (A.Y) - FAC1, Unterprogramm
E97F	FAC1 = (A,Y) * FAC1, Unterprogramm
EA66	FAC1 = (A,Y) / FAC1, Unterprogramm
EB53	FAC2 nach FAC1 übertragen, Unterprogramm
EAF9	(A,Y) nach FAC1 übertragen, Unterprogramm
D412	Fehlermeldungen behandeln
F280	FLASH, bewirkt blinkende Bildschirmausgabe
E354	FN auswerten, Unterprogramm
D766	FOR, Schleife definieren
E2DE	FRE, bringt freien Speicherplatz
E484	Garbage Collection Routine, Unterprogramm
DBA0	GET, Eingabe eines Zeichens
D921	GOSUB, Basic-Unterprogramm aufrufen
D93E	GOTO, unbedingter Sprung
F2E9	GOTO nach ONERR, zur Fehlerbehandlungsroutine
F390	GR, LoRes-Grafikmodus wählen
F457	Grafikpunkt in HiRes-Grafik setzen, Unterprogramm
F6E9	HCOLOR, Farbe für HiRes-Grafik wählen
F3E3	HGR, HiRes-Grafikmodus, Page1 wählen
F3D8	HGR2, HiRes-Grafikmodus, Page2 wählen
F286	HIMEM:, Ende des benutzbaren RAM definieren

F232	HLIN, horizontale Linie in LoRes-Grafik zeichnen
D52C	hole eine Befehlszeile, Unterprogramm
D553	hole einzelnes Eingabezeichen, Unterprogramm
DE60	hole Operand aus Programmtext, Unterprogramm
F72D	hole Parameter für DRAW/XDRAW, Unterprogramm
F209	hole Parameter für HLIN/VLIN, Unterprogramm
F6B9	hole Parameter für HPLOT, DRAW/XDRAW, Unterprogramm
F1EC	hole Parameter für LoRes-Grafik, Unterprogramm
DED5	hole Variablenwert, Unterprogramm
DA0C	hole Zeilennummer aus Programmtext, Unterprogramm
E6F5	hole 1-Byte-Integer aus Programmtext, Unterprogramm
E746	hole 2-Byte-Integer und 1-Byte-Integer, Unterprogramm
F6FE	HPLOT, Punkt oder Linie in HiRes-Grafik zeichnen
F7E7	HTAB, Spaltentabulator
D9C9	IF, bedingte Anweisung
DBB2	INPUT, Eingaberoutine
EC23	INT, Ganzzahlfunktion
E2F2	Integer (A,Y) als FP-Konstante in FAC1 bringen, Unterprogramm
D7D2	Interpreter-Hauptschleife
F277	INVERSE, bewirkt inverse Bildschirmausgabe
F1DE	IN#, Eingabekanal wählen
F128	Kaltstart
E65A	LEFT$, bringt linken Teilstring
E6D6	LEN, bringt Stringlänge
DA46	LET, Variablenzuweisung
F53A	Linie in HiRes-Grafik zeichnen, Unterprogramm
D6A5	LIST, Programm auflisten
D8C9	LOAD, Programm von Cassette laden
E941	LOG, natürlicher Logarithmus
F2A6	LOMEM:, Anfang des Variablenbereichs wählen
E691	MID$, bringt mittleren Teilstring
D8F0	Monitorregister für Längendaten setzen, Unterprogramm
D901	Monitorregister für Programmtext setzen, Unterprogramm
F7BC	Monitorregister für STORE/RECALL setzen, Unterprogramm
E2AD	Multiplikation (16Bit), Unterprogramm
D649	NEW, Programm löschen
DCF9	NEXT, Schleifenwiederholung
F273	NORMAL, normale Bildschirmausgabe wählen
DE98	NOT, logische Negation
F26F	NOTRACE, TRACE-Modus abschalten
D9EC	ON, bedingter Sprung
F2CB	ONERR, Fehlerbehandlungsroutine definieren
DF4F	OR, logische Oder-Verknüpfung
DFCD	PDL, Stellung der Paddles abfragen
E764	PEEK, Inhalt einer Speicherzelle abfragen
F225	PLOT, Punkt in LoRes-Grafik zeichnen
E77B	POKE, Inhalt einer Speicherzelle verändern
EF72	Polynom auswerten, Unterprogramm
EF5C	Polynom (ungerade) auswerten, Unterprogramm
D96B	POP, RETURN-Adresse vom Stack entfernen
E2FF	POS, bringt horizontale Cursorposition
DAD5	PRINT, Ausgaberoutine
D45C	Programmzeile in Programmtext einfügen
D559	Programmzeile kodieren (Token ersetzen Keywords), Unterprogramm
D3E3	prüfe freien Speicherplatz, Unterprogramm
DD6A	prüfe, ob Ausdruck numerische ist, Unterprogramm
DD6C	prüfe, ob Stringausdruck, Unterprogramm
D3D6	prüfe Platz auf Stack, Unterprogramm
E07D	prüfe Zeichen auf Buchstabe, Unterprogramm
F1E5	PR#, Ausgabekanal wählen
F465	RAM-Adresse für horizontalen Nachbarpunkt bestimmen, Unterprogramm

F4D3	RAM-Adresse für vertikalen Nachbarpunkt bestimmen, Unterprogramm
F5CB	RAM-Adresse in XY-Koordinaten umwandeln, Unterprogramm
DBE2	READ, Daten aus Programmtext holen
F3BC	RECALL, Feldvariable von Cassette laden
D9DC	REM, Bemerkung im Programmtext
D849	RESTORE, DATA-Zeiger auf Anfang zurücksetzen
F318	RESUME, Programmausführung nach Fehlerbehandlung wieder aufnehmen
D96B	RETURN, Rücksprung vom Unterprogramm
E686	RIGHT$, bringt rechten Teilstring
EFAE	RND, Zufallszahl erzeugen
F721	ROT=, Drehwinkel für Shape wählen
D912	RUN, Programm ausführen
D8B0	SAVE, Programm auf Cassette speichern
F727	SCALE=, Vergrößerungsfaktor für Shape wählen
DEF9	SCRN, bringt Farbkode des angesprochenen LoRes-Grafikpunkts
EB90	SGN, Vorzeichenfunktion
F4B3	Shape-Anweisung für DRAW durchführen, Unterprogramm
F49C	Shape-Anweisung für XDRAW durchführen, Unterprogramm
F775	SHLOAD, Shape von Cassette laden
EFF1	SIN, Sinus-Funktion
F262	SPEED, Ausführungsgeschwindigkeit wählen
D39A	Speicherblock verschieben, Unterprogramm
EE8D	SQR, Wurzelfunktion
D683	Stack initialisieren, Unterprogramm
D86E	STOP, Programm unterbrechen
F39F	STORE, Feldvariable auf Cassette speichern
E5FD	String, der nicht mehr benötigt wird, entfernen; Unterprogramm
E5D4	String kopieren, Unterprogramm
E452	Stringbereich für neuen String vorbereiten, Unterprogramm
E3E7	Stringdeskriptor bestimmen, Unterprogramm
DA7A	Stringdeskriptor in Variable eintragen
E597	Strings verketten
E3C5	STR$, Zahl in String umwandeln
F7D9	suche Feldvariable für STORE/RECALL, Unterprogramm
D365	Suche FOR/NEXT-Parameter im Stack, Unterprogramm
D9A3	suche nächstes Trennzeichen, Unterprogramm
D61A	suche Programmzeile, Unterprogramm
DFE3	suche Variable, Unterprogramm
D9A5	suche Zeilenende, Unterprogramm
F03A	TAN, Tangens-Funktion
F399	TEXT, Bildschirmausgabe auf Textmodus setzen
F26D	TRACE, Zeilennummern ausgeführter Befehle auflisten
E707	VAL, String in Zahl umwandeln
DF65	Vergleichsoperation durchführen, Unterprogramm
F241	VLIN, vertikale Linie in LoRes-Grafik zeichnen
EB82	Vorzeichen von FAC1 testen, Unterprogramm
F256	VTAB, Zeilentabulator
E784	WAIT, warten bis angegebene Speicherzelle Sollwert hat
D43C	Warmstart
F76F	XDRAW, Shape zeichnen (dabei jeden Punkt invertieren)
F661	XDRAW, Unterprogramm
F411	XY-Koordinaten in RAM-Adresse umrechnen, Unterprogramm
EC4A	Zahlenstring in FP-Konstante umwandeln, Unterprogramm

Änderungen bei den Modellen IIe und IIc

Beim Apple IIe wurden zwar Änderungen im Monitor-ROM vorgenommen, im Bereich des Basic-Interpreters jedoch nicht. Dies bedeutet, dass der vorliegende Kommentar uneingeschränkt für den IIe benutzt werden kann.

Beim IIc jedoch wurden auch im Basic-Interpreter einige Stellen geändert. Da der IIc schon einen Diskdrive eingebaut hat, wurde auf ein Cassettenbetriebssystem ganz verzichtet. Dadurch werden sämtliche Cassettenroutinen überflüssig. Dafür wurden vor allem LoRes-Grafikroutinen eingefügt, welche die LoRes-Befehle (HLIN etc.) auch im 80-Zeichen-Modus erlauben. Nachfolgend sind sämtliche geänderten Stellen aufgeführt.

Änderungen in der Sprungtabelle:

```
D034-   F4 03       SHLOAD  (zeigt nun auf &-Vektor)
...
D04E-   F4 03       RECALL
D050-   F4 03       STORE
...
D06C-   F4 03       LOAD
D06E-   F4 03       SAVE
```

Änderungen in der Keyword-Kodierungsroutine:

```
D56D-   JSR   $D8B5     hole Zeichen aus Tastaturpuffer,akzeptiere
...                     Kleinbuchstaben
D5A8-   JSR   $D8B5     dto
...
D5BE-   JSR   $D8B0     dto
...
D5E9-   JSR   $D8B5     dto
...
D60B-   JSR   $D8C3     dto
```

Änderungen in der LIST-Routine:

```
D6F9-   JSR   $D8D3     drucke Leerzeichen, dann die Zeilennummer
...
D705-   JSR   $D8DD     prüfe Cursorspalte (max.33/73 bei 40/80 Zeichen)
D708-   NOP
```

Zeichen aus dem Tastaturpuffer holen

```
D8B0-   LDA   $0201,X   Originalzeichen aus Tastaturpuffer holen
D8B3-   BPL   $D8C6     springe immer
D8B5-   LDA   $0E       REM-Flag
D8B7-   BEQ   $D8CF     REM-Status, Originalzeichen holen
D8B9-   CMP   #$22      Text-Status?
D8BB-   BEQ   $D8CF     ja, Originalzeichen holen
D8BD-   LDA   $13       DATA-Flag
D8BF-   CMP   #$49      gesetzt?
D8C1-   BEQ   $D8CF     ja, Originalzeichen holen
D8C3-   LDA   $0200,X   hole Zeichen zur Umwandlung in Grossbuchstaben
D8C6-   PHP             rette Z-Flag
D8C7-   CMP   #$61      Ascii für "a"
D8C9-   BCC   $D8CD     kein Kleinbuchstabe
D8CB-   AND   #$5F      sonst in Grossbuchstaben umwandeln
D8CD-   PLP             Z-Flag wiederherstellen
D8CE-   RTS
D8CF-   LDA   $0200,X   Zeichen von Tastaturpuffer holen
```

```
D8D2-   RTS

Hilfsroutinen für LIST-Routine

D8D3-   PHA                 Zeilennummer, HByte  retten
D8D4-   LDA   #$20          Ascii für Leerzeichen
D8D6-   JSR   $DB5C         drucken
D8D9-   PLA                 gerettete Zeilennummer, HByte
D8DA-   JSR   $ED24         Zeilennummer drucken, fertig

Listformat-Prüfung
D8DD-   LDA   $24           Cursorspalte
D8DF-   CMP   #$21          mit 33 vergleichen
D8E1-   BIT   $C01F         Bit7=1 falls 80-Zeichen-Modus
D8E4-   BPL   $D8EB         fertig
D8E6-   LDA   $057B         Cursorspalte bei 80-Zeichen-Modus
D8E9-   CMP   #$49          mit 73 vergleichen
D8EB-   RTS
---------------------

LoRes-Grafikmodus initialisieren
D8EC-   LDA   $C050         setze GRAPHICS
D8EF-   JSR   $D8F7         lösche LoRes-Grafikseite
D8F2-   LDA   #$14          oberer Textfensterrand in Zeile 20
D8F4-   JMP   $FB4B         Monitorroutine SETWND setzt Textfenster

LoRes-Grafikseite löschen
D8F7-   LDY   #$27          löschen bis Grafikzeile 39 (d.h. bis Textzeile 19)
D8F9-   STY   $2D
D8FB-   JSR   $F3CB         prüfe ob Double-LoRes (also 80-Zeichen-Format)
D8FE-   LDA   #$27          rechter Rand = Spalte 39, falls Single-LoRes
D900-   BCC   $D903         Single-LoRes Modus
D902-   ROL                 sonst rechter Rand = Spalte 79
D903-   TAY                 als Spaltenzeiger
D904-   LDA   #$00          COLOR-Byte für schwarz
D906-   STA   $30           eintragen
D908-   JSR   $F78B         LoRes-Spalte löschen
D90B-   DEY                 nächste Spalte
D90C-   BPL   $D904         bis Spalte 0 erreicht ist
D90E-   RTS
D90F-   BRK
D910-   BRK
D911-   BRK
---------------------

Änderungen in der PRINT-Routine
DADB-   BEQ   $DB19         neuer Einsprung in TAB / SPC -Routine
...
DAE0-   BEQ   $DB19         dto

Cursor auf nächsten Tabulator setzen
DB03-   JSR   $D8DD         Cursorspalte prüfen
DB06-   BMI   $DB11         kleiner als 73  (80-Zeichen-Modus)
DB08-   CMP   #$18          sonst mit 24 vergleichen
DB0A-   BCC   $DB11         kleiner, also in selber Zeile bleiben
DB0C-   JSR   $DAFB         sonst CR ausgeben (neue Zeile)
DB0F-   BNE   $DB2F         weiter mit neuem Zeichen (immer)
DB11-   ADC   #$10          nächstes Vielfaches von 16 bestimmen
DB13-   AND   #$F0
DB15-   TAX
```

```
DB16-   SEC
DB17-   BCS   $DB25      Cursor setzen


TAB / SPC  ausführen
DB19-   PHP              C retten  ( =0 bei SPC, =1 bei TAB )
DB1A-   JSR   $E6F5      hole 8-Bit-Integer aus Programmtext --> X
DB1D-   CMP   #$29       folgendes Zeichen = ")" ?
DB1F-   BNE   $DB83      nein, SYNTAX ERROR
DB21-   PLP              hole C-Flag
DB22-   BCC   $DB2B      SPC, Leerzeichen drucken
DB24-   DEX              TAB-Wert
DB25-   JSR   $F7CB      aktuelle Cursorspalte subtrahieren
DB28-   ....   von hier ab unverändert   ............
---------------------

Änderung in SCRN - Routine
DF02-   JSR   $F7A6      bringt Farbcode des LoRes-Punktes
---------------------

Änderungen in LoRes-Parameter - Auswertungsroutine
F1EF-   CPX   #$50       Limit = 80  (statt 48)
F1FD-   CPX   #$50       dto
F220-   CPX   #$50       dto
---------------------

Änderungen in der PLOT - Routine
F225-   JSR   $F1EC      PLOT-Parameter aus Programmtext holen
F228-   LDY   $F0        X-Koordinate
F22A-   JSR   $F775      muß kleiner als 40 bzw 80 sein (je nach Modus)
F22D-   TXA              Y-Koordinate
F22E-   JMP   $F3F9      setze LoRes-Punkt
F231-   ???
---------------------

Änderungen in der  HLIN - Routine
F232-   JSR   $F209      hole 3 Parameter aus Programmtext
F235-   LDY   $2C        X-Koordinate des rechten Endes der Linie
F237-   JSR   $F775      muß kleiner als 40 bzw 80 sein (je nach Modus)
F23A-   CPX   #$30       Y-Koordinate mit 48 vergleichen
F23C-   BCS   $F206      zu groß, -->  ILLEGAL QUANTITY ERROR
F23E-   JMP   $F796      zur Darstellungsroutine

Änderungen in der  VLIN - Routine
F241-   JSR   $F209      hole 3 Parameter aus Programmtext
F244-   TXA              X-Koordinate
F245-   TAY
F246-   JSR   $F775      muß kleiner als 40 bzw 80 sein (je nach Modus)
F249-   LDA   $F0        Y-Koordinate des oberen Endes der Linie
F24B-   JMP   $F783      zur Darstellungsroutine
F24E-   BRK
---------------------

Änderungen in der GR - Routine
F390-   LDA   $C056      setze LORES
F393-   LDA   $C053      setze MIX
F396-   JMP   $D8EC      zur LoRes-Initialisierungsroutine
---------------------

LoRes-Punkt setzen
I: Accu = Y-Koordinate,  Y-Reg. = X-Koordinate
F39F-   LSR              LoRes-Zeile halbieren --> Textzeile
```

```
F3A0-   PHP              C retten  (C=1 bei ungeraden Lores-Zeilen)
F3A1-   JSR   $F847      Monitorroutine GBASCALC bringt Zeilenadresse
F3A4-   PLP              hole C
F3A5-   LDA   #$0F       Maske für gerade LoRes-Zeilen
F3A7-   BCC   $F3AB      ok, war gerade
F3A9-   ADC   #$E0       #$E1 addieren (C=1!), ergibt #$F0 als Maske
F3AB-   STA   $2E        Maske eintragen
F3AD-   PHY              Y auf Stack retten
F3AE-   JSR   $F7BB      Page2 wählen, falls Double-LoRes u. ungerade Spalte
F3B1-   BCC              Page1 aktiv
F3B3-   PHX              X auf Stack retten
F3B4-   LDA   $30        COLOR-Byte
F3B6-   TAX              in X merken
F3B7-   LSR              Bit0 --> C
F3B8-   TXA              COLOR-Byte holen
F3B9-   ROR              und korrigieren
F3BA-   SEC              C=1, da Page2 - Status
F3BB-   STA   $30        COLOR-Byte eintragen
F3BD-   JSR   $F80E      Monitorroutine PLOT1 setzt LoRes-Punkt
F3C0-   BCC   $F3C9      Page1 - Status, fertig
F3C2-   LDA   $C054      auf Page1 zurückschalten
F3C5-   STX   $30        altes COLOR-Byte wiederherstellen
F3C7-   PLX              gerettetes X zurück
F3C8-   CLC
F3C9-   PLY              gerettetes Y zurück
F3CA-   RTS
--------------------

Überprüfung der Screen-Softswitches
F3CB-   LDA   $C079      AN3  (Bit7 = 0 bei  Double on)
F3CE-   EOR   #$80
F3D0-   AND   $C018      und 80-STORE
F3D3-   AND   $C01F      und 80-COLUMN, --> Bit7=1 falls alles erfüllt ist
F3D6-   ASL              Bit7 --> C
F3D7-   RTS
--------------------

prüfe, ob Spaltennummer zulässig
F775-   JSR   $F3CB      Double-LoRes aktiv?
F778-   BCS   $F77E      ja
F77A-   CPY   $#$28      Spalte kleiner als 40?
F77C-   BCS   $F73F      nein, --> ILLEGAL QUANTITY ERROR
F77E-   CPY   #$50       Spalte kleiner als 80?
F780-   BCS   $F73F      nein, --> ILLEGAL QUANTITY ERROR
F782-   RTS
--------------------

Darstellungsroutine für  VLIN
F783-   PHA              Y-Koordinate des oberen Endes der Linie retten
F784-   LDA   $2D        Y-Koordinate des unteren Endes der Linie
F786-   CMP   #$30       kleiner als 48?
F788-   PLA
F789-   BCS   $F73F      nein, --> ILLEGAL QUANTITY ERROR
F78B-   PHA              Y-Koordinate merken
F78C-   JSR   $F39F      LoRes-Punkt setzen
F78F-   PLA              aktuelle Y-Koordinate holen
F790-   CMP   $2D        unteres Ende der Linie erreicht ?
F792-   INC              Koordinate für unteren Nachbarpunkt
F793-   BCC   $F78B      noch nicht fertig, weiter
F795-   RTS

Darstellungsroutine für  HLIN
F796-   TXA              Y-Koordinate
```

```
F797-   LDY   $F0       X-Koordinate des linken Endes der Linie
F799-   JSR   $F39F     LoRes-Punkt setzen
F79C-   CPY   $2C       rechtes Ende der Linie erreicht ?
F79E-   BCS   $F795     ja, fertig
F7A0-   INY             nächster Punkt
F7A1-   JSR   $F3AD     LoRes-Punkt setzen
F7A4-   BRA   $F79C     weiter bis Linie fertig
--------------------

Hilfsroutine für  SCRN
F7A6-   PHA             Y-Koordinate retten
F7A7-   JSR   $F7BB     richtige LoRes-Page adressieren
F7AA-   PLA
F7AB-   PHP             C retten  (C=1 falls Page2 gewählt)
F7AC-   JSR   $F871     Monitorroutine SCRN bringt den Farbcode
F7AF-   PLP
F7B0-   BCC   $F7BA     Page1 war gewählt, fertig
F7B2-   STA   $C054     zurück zu Page1
F7B5-   CMP   #$08      Farbcode korrigieren
F7B7-   ASL
F7B8-   AND   #$0F      gültige Bits maskieren
F7BA-   RTS
--------------------

gültige LoRes-Page auswählen
F7BB-   JSR   $F3CB     Screen-Softswitches prüfen
F7BE-   BCC   $F7CA     nicht Double-LoRes
F7C0-   TYA             X-Koordinate
F7C1-   EOR   #$01      Bit0 invertieren
F7C3-   LSR             Koordinate halbieren, Bit0 --> C
F7C4-   TAY             korrigierte Koordinate zurück nach Y
F7C5-   BCC   $F7CA     X-Koordinate war gerade, fertig
F7C7-   LDA   $C055     sonst Page2 anwählen
F7CA-   RTS
--------------------

Hilfsroutine für PRINT
F7CB-   TXA             Tabulatorspalte
F7CC-   BIT   $C01F     80-Zeichen-Modus ?
F7CF-   BMI   $F7E3     ja
F7D1-   BIT   $2485     --> F7D2-  STA  $24
F7D4-   SEC
F7D5-   TXA
F7D6-   SBC   $24       Cursorspalte (40Z-Modus) subtrahieren
F7D8-   RTS
--------------------

Suche Feldvariable (z.B. für Store / Recall)
F7D9-   LDA   #$40
F7DB-   STA   $14       STORE-Flag setzen
F7DD-   JSR   $DFE3     angegebene Feldvariable suchen
F7E0-   STZ   $14       STORE-Flag löschen
F7E2-   RTS
--------------------

Rest von F7CB ff
F7E3-   SBC   $057B     Cursorspalte (80Z-Modus) subtrahieren
F7E6-   RTS


Änderungen in der  HTAB - Routine
F7E7-   JSR   $E6F8     1-Byte-Integer aus Programmtext holen --> X
```

```
F7EA-   DEX
F7EB-   LDA   #$28      40
F7ED-   CMP   $21       mit der Breite des Bildschirmfensters vergleichen
F7EF-   BCS   $F7F3     Breite =< 40
F7F1-   LDA   $21       Breite
F7F3-   JSR   $F7D2     HTAB-Position minus Accu
F7F6-   STX   $24       vorläufige Cursorspalte
F7F8-   BCC   $F7D8     HTAB-Position < rechter Bildschirmrand, fertig
F7FA-   TAX             sonst noch verbliebenen Abstand merken
F7FB-   JSR   $DAFB     Zeilenvorschub
F7FE-   BRA   $F7EB     weiter
--------------------
```